Hybrid Meta-heuristic Algorithms for Static and Dynamic Job Scheduling in Grid Computing

By

Muhanad Tahrir Younis

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

in the

School of Computer Science and Informatics
De Montfort University
Leicester, UK

September 2018

*I would like to dedicate this work to my lovely daughter **Roza**, without you this thesis would not have been completed. You have been a constant source of inspiration during the most difficult and challenging time. I love you to the next galaxy and back . . .*

Declaration

The content of this submission was undertaken in the School of Computer Science and Informatics, De Montfort University, and supervised by Professor Shengxiang Yang and Dr. Benjamin Passow during the period of registration. I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 58,900 words including appendix, bibliography, figures, tables and equations. Part of the research work presented in this submission has been published in the papers listed in Chapter 1.

By

Muhanad Tahrir Younis
September 2018

Acknowledgements

I would like to express my special appreciation and thanks to my first supervisor Professor Shengxiang Yang, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advice on both research as well as on my career have been priceless. I would also like to thank my second supervisor, Dr Benjamin Passow, for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

A special thanks to my family. Words cannot express how grateful I am to my parents, your prayers for me were what sustained me thus far, and to my brothers and sister for supporting me spiritually throughout writing this thesis and my life in general. I would also like to express my appreciations to my beloved uncle Ra'ad who was always my support in the moments when there was no one.

I am also very grateful to all my friends, especially Nahle and her adorable daughters Fatima and Sara, Phuong, Luong, Liane, Ray and his wife, John and his wife, Dalton and his family, Derek and many others from Globe Cafe, for their support and encouragement during my PhD journey.

Special thanks go to my colleagues and my fellow office-mates for the stimulating discussions and for all the fun we have had in the last four years: Dr Shouyong Jiang, Dr Jayne Eaton, Conor Fahy, Matthew Fox, Olukemi Olowofoyeku, Zedong Zheng, and Qingyang Zhang. Without them, the research would not have been such a pleasant and enjoyable experience in my life. Also I thank my colleagues in the Department of Computer Science at Al-Mustansiriya University in Baghdad. In particular, I am grateful to Professor Saad Alsaad for his endless support.

Finally, it would not have been possible to carry out this research work without the kind sponsorship of the Iraqi government, represented by the Ministry of Higher Education and Scientific Research, the Iraqi Cultural Attaché in London, and Al-Mustansiriya University in Baghdad, and so I am grateful to extend my sincere thanks and appreciations to them.

Abstract

The term 'grid computing' is used to describe an infrastructure that connects geographically distributed computers and heterogeneous platforms owned by multiple organizations allowing their computational power, storage capabilities and other resources to be selected and shared. Allocating jobs to computational grid resources in an efficient manner is one of the main challenges facing any grid computing system; this allocation is called job scheduling in grid computing. This thesis studies the application of hybrid meta-heuristics to the job scheduling problem in grid computing, which is recognized as being one of the most important and challenging issues in grid computing environments. Similar to job scheduling in traditional computing systems, this allocation is known to be an NP-hard problem. Meta-heuristic approaches such as the Genetic Algorithm (GA), Variable Neighbourhood Search (VNS) and Ant Colony Optimisation (ACO) have all proven their effectiveness in solving different scheduling problems. However, hybridising two or more meta-heuristics shows better performance than applying a stand-alone approach. The new high level meta-heuristic will inherit the best features of the hybridised algorithms, increasing the chances of skipping away from local minima, and hence enhancing the overall performance. In this thesis, the application of VNS for the job scheduling problem in grid computing is introduced. Four new neighbourhood structures, together with a modified local search, are proposed. The proposed VNS is hybridised using two meta-heuristic methods, namely GA and ACO, in loosely and strongly coupled fashions, yielding four new sequential hybrid meta-heuristic algorithms for the problem of static and dynamic single-objective independent batch job scheduling in grid computing. For the static version of the problem, several experiments were carried out to analyse the performance of the proposed schedulers in terms of minimising the makespan using well known benchmarks. The experiments show that the proposed schedulers achieved impressive results compared to other traditional, heuristic and meta-heuristic approaches selected from the bibliography. To model the dynamic version of the problem, a simple simulator, which uses the rescheduling technique, is designed and new problem instances are generated, by using a well-known methodology, to evaluate the performance of the proposed hybrid schedulers. The experimental results show that the use of rescheduling provides significant improvements in terms of the makespan compared to other non-rescheduling approaches.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Grid Computing is an infrastructure that connects multiple heterogeneous and autonomous resources such as databases, computers and servers, from different domains – which can be geographically distributed worldwide – to perform various complex tasks. Depending upon their availability, performance and users' requirements, this infrastructure allows the dynamic sharing of various resources and thus creates a virtual supercomputer. Grid computing was mainly developed to fulfil the significantly increasing requirements for high computing power being demanded by various organizations and the scientific computing community. From this perspective, it has been used as a utility for diverse applications that require intensive computing power from commercial and non-commercial clients [58].

The early definitions of grid computing systems were introduced in [56] [59], and since then many developments have taken place in terms of both grid infrastructure and middleware, resulting in a better understanding of grid issues. One of the main challenges in a computational grid is how to efficiently map jobs, also called tasks or applications, to grid resources and hence utilize geographically distributed computers which are connected through heterogeneous environments in an efficient, reliable and secure manner. This mapping is called job scheduling in grid computing. Similar to job scheduling in traditional computing systems, this mapping is known to be an NP-hard problem [161]. However, it is more complicated in grid computing due to its complex, dynamic nature, high degree of job and resource heterogeneity, problem size, and other factors such as existing local schedulers and policies [25].

There are several ways to classify scheduling problems in grid computing. One way includes the number of objectives to be optimized, in which the problem can be categorized into single or multi-objective. Another way is according to the processing mode, by which they can be classified into immediate or batch. The immediate scheduling refers to the type of mapping in which jobs are scheduled once they arrive to the system, while jobs are collected in a group or a batch, then this group is scheduled in the batch mode. Therefore, better schedules are expected as this mode takes the advantages of knowing the

characteristics of all submitted jobs and the available resources, which helps in making decisions about the job-resource mapping process [88].

Moreover, the interrelations between jobs can be used to classify scheduling problems into independent or dependent. In the former category, jobs are not related to each other, that is, inter-job dependencies are not available. In the latter type, jobs cannot be divided as they should be processed using a predefined sequence, which means inter-job dependencies must be considered [55]. The independent job scheduling suits most of the characteristics of distributed heterogeneous environments such as grid computing systems. This is mainly due to the nature of their users, as different independent users submit their jobs and applications to be processed by these environments. Additionally, the importance of the independent job scheduling arises in various realistic applications. Examples include those applications which use the SPMD (Single Program, Multiple Data) technique, such as data mining and image processing applications. Furthermore, this type of scheduling is useful for applications that can be divided into independent parts such as Monte-Carlo simulations and parameter sweep applications [25] [85] [117].

Scheduling problems in grid computing may also be categorised according the type of environment, by which they can be static or dynamic. In the first category, the necessary information about jobs and resources are available in advance. These information will not be changed during the mapping process. In addition, no jobs are expected to arrive at the system after the allocation is performed [118]. This type of scheduling is useful in many different applications and domains such as predictive analyses, distributed computing systems requirements analyses and to study the behaviour of a dynamic scheduler in terms of resources selection [19] [135]. On the other hand, in the second category which includes the dynamic scheduling, jobs and resources can be added and removed to the system at runtime. This provides an efficient way to cope with any unpredicted events such as resource failure. A scheduler of this type uses actual information instead of estimations to assign jobs to resources [163].

In this thesis, the static batch independent job scheduling problem version in grid computing is considered. Different hybrid meta-heuristic methods are proposed to tackle this problem in terms of minimising a single objective, the makespan. Furthermore, to tackle the dynamic version of the problem, the rescheduling strategy will be added to these methods.

1.1 Motivation

The problem of job scheduling in grid computing has been addressed using different approaches such as simple queuing algorithms, deterministic heuristic algorithms and meta-heuristic algorithms. However, to effectively deal with its complexity, meta-heuristic

algorithms are preferred [85]. Meta-heuristic algorithms are well-known approaches which have been applied effectively to a wide range of NP-hard problems. In fact, these algorithms are considered the best candidate in practice to cope with the complexity of job scheduling in a computational grid, and accordingly several algorithms have been suggested [124].

Meta-heuristic approaches such as the Genetic Algorithm (GA), Tabu Search (TS), Particle Swarm Optimisation (PSO), Variable Neighbourhood Search (VNS) and Ant Colony Optimisation (ACO) have all proven their effectiveness in solving different scheduling problems. However, hybridising two or more meta-heuristics shows better performance than applying a stand-alone approach [10]. The new high level meta-heuristic will inherit the best features of the hybridised algorithms, increasing the chances of skipping away from local minima, and hence enhancing the overall performance [170].

Four main issues that must be examined in order to design a new hybrid algorithm, namely the number of algorithms to hybridize, the type of algorithms to hybridize, the execution mode (sequential or parallel) and the hybridization type [169]. Meta-heuristic methods can be hybridized in two ways, either as loosely coupled or strongly coupled [167]. The former refers to the case in which the hybridized algorithms preserve their identity by running as a chain of executions in which the output of the first algorithm will be used by the second, and so on; the final solution will be the output of the last algorithm. The latter refers to the type of hybridization in which the inner procedures of the hybridized algorithms are interchanged in such a way as one of the methods acts as the main algorithm which, during its execution, calls other methods to act as supporting algorithms [170].

The literature shows that the use of hybrid meta-heuristics, such as ACO [135], GA [25] and VNS [138], for the job scheduling problem in grid computing provided impressive results; however, almost all the proposed meta-heuristic hybrid algorithms in the literature were loosely coupled. Moreover, the VNS algorithm is considered a framework for building heuristic algorithms which has been used efficiently and effectively in various optimisation problems [72]. However, for the job scheduling problem in grid computing, VNS was used as a stand-alone algorithm in all the available works [138]. These observations pave the way to the examination of the hybridisation of ACO and GA with VNS in a loosely and strongly coupled fashion and compare the performance of such couplings with existing methods.

1.2 Objectives

In this thesis, the application of VNS for the job scheduling problem in grid computing is introduced. Four new neighbourhood structures, together with a modified local search, are proposed. The proposed VNS is hybridised using two meta-heuristic methods in

loosely and strongly coupled fashions, yielding four new sequential hybrid meta-heuristic algorithms for the problem of static single-objective independent batch job scheduling in grid computing. The first algorithm, called ACO+VNS, combines ACO and VNS, in which the former works first and whose output is further refined by the latter algorithm, while the second algorithm, called GA+VNS, integrates GA and VNS in the same manner. The third algorithm, called ACO(VNS), combines ACO and VNS in which the former acts as the primary algorithm which calls during its execution the latter as a supporting algorithm, while the forth algorithm, called GA(VNS), integrates GA and VNS in the same manner. Using the Expected Time to complete (ETC) simulation model, several experiments have been carried out to evaluate the performance of the proposed methods in terms of minimising the makespan. Three different well known datasets were used for this purpose rather than generating a special dataset so that we could easily make a fair comparison to other state-of-the-art methods. Moreover, these four hybrid schedulers have been applied to the dynamic mode of the problem. A well-known technique called rescheduling was used to introduce dynamism to the problem. The performance of the proposed hybrid dynamic meta-heuristic algorithms was evaluated by using a benchmark which has been especially created for the dynamic job scheduling problem.

1.3 Contributions

The main contributions of this work are:

1. To analyse the use of various meta-heuristic approaches for solving the job scheduling problem in grid computing.
2. To develop a new VNS meta-heuristic for the job scheduling problem in grid computing, which uses some effective and carefully designed neighbourhood structures and a powerful local search to explore different regions on the state space of the problem.
3. To design and implement several novel hybrid meta-heuristic schedulers, in loosely and strongly coupled fashions, that uses the newly proposed VNS to produce high quality schedules in a reasonable time.
4. To provide the literature with new state-of-the-art sequential hybrid algorithms for job scheduling in grid computing that could serve as a reference in the field.
5. To generate new problem instances, by using a well-known methodology, which model dynamic job scheduling in grid computing.
6. To introduce a dynamic scenario which uses the rescheduling technique to simulate the dynamic job scheduling problem.

7. To use the introduced dynamic scenario to evaluate the performance of the proposed hybrid schedulers.

1.4 Thesis structure

The reminder of the manuscript is organized as follows. Chapter 2 describes the ideas and generic concepts that form the basis for the hybrid meta-heuristic algorithms proposed in this thesis. The chapter begins by defining combinatorial optimisation problems. Then, an overview of heuristic, meta-heuristic and hybrid meta-heuristic algorithms is presented. The chapter continues by introducing detailed descriptions of some meta-heuristic algorithms, more precisely VNS, ACO and GA, which have been used to solve the job scheduling problem in grid computing in this research.

Chapter 3 introduces an overview of grid computing and its basic components. Additionally, the chapter presents the job scheduling problem description and the simulation model that mimics the assignment of jobs to resources. Furthermore, the main methods described in the literature to generate problem instances are also explained. Finally, it presents a comprehensive review of the static and dynamic heuristic and meta-heuristic approaches that were used to tackle the job scheduling problem in heterogeneous environments such as grid computing.

Chapter 4 introduces the use of VNS for job scheduling in grid computing. Four new neighbourhood structures, together with a modified local search, are proposed. The proposed VNS is hybridized with two meta-heuristic methods in a loosely and strongly coupled fashions, yielding new hybrid meta-heuristic algorithms by which to consider the job scheduling problem in grid computing.

Chapter 5 discusses the experimental results of applying the four proposed hybrid meta-heuristic algorithms for the static independent job scheduling in grid computing. The development language and the other measures used to report the achieved results are also described in this chapter. Moreover, it describes the essential experiments carried out to select the best parameters for each proposed method to best optimize its performance.

Chapter 6 introduces the application of hybrid meta-heuristic algorithms for solving the dynamic job scheduling problem in grid computing in terms of minimising the makespan. A version of the dynamic job scheduling problem in grid computing in which blocks of independent jobs arrive to the grid system at different arrival time, was considered. To solve this problem, the rescheduling strategy, which involves several calls of the job scheduler at various intervals of time, is employed. The dynamic simulation model and how to apply rescheduling are also explained in this chapter. Moreover, the chapter studies the performance of the proposed meta-heuristic algorithms by using a benchmark which has been especially created for the dynamic job scheduling problem.

Finally, our conclusions along with the possible future works are drawn in the seventh and final chapter.

1.5 List of related publications by the author

Part of the research work presented in this thesis has been published in the following papers:

1. Muhanad Tahrir Younis, Shengxiang Yang, and Benjamin Passow. Meta-heuristically seeded genetic algorithm for independent job scheduling in grid computing. In *European Conference on the Applications of Evolutionary Computation*, pages 177–189. Springer, 2017.

2. Muhanad Tahrir Younis and Shengxiang Yang. A genetic algorithm for independent job scheduling in grid computing. *MENDEL Soft Computing Journal*, 23(01):65–72, 2017.

3. Muhanad Tahrir Younis and Shengxiang Yang. Hybrid meta-heuristic algorithms for independent job scheduling in grid computing. *Applied Soft Computing*, 2018.

4. Muhanad Tahrir Younis, Shengxiang Yang, and Benjamin N Passow. A loosely coupled hybrid meta-heuristic algorithm for the static independent task scheduling problem in grid computing. In *IEEE World Congress on Computational Intelligence*, pages 1746–1753. IEEE Press, 2018.

Chapter 2

Hybrid Meta-Heuristic Algorithms

The aim of this chapter is to describe the ideas and generic concepts that form the basis for the hybrid meta-heuristic algorithms proposed in this thesis. The chapter begins by defining combinatorial optimisation problems. Then, an overview of heuristic, meta-heuristic and hybrid meta-heuristic algorithms is presented. The chapter continues by introducing detailed descriptions of some meta-heuristic algorithms, more precisely VNS, ACO and GA, which will be used to solve the job scheduling problem in grid computing in this thesis. Finally, a summary of the topics covered in this chapter is provided.

2.1 Combinatorial optimisation problems

The branch of mathematics and computational science that investigates the various algorithms, techniques and methods that are proposed to find the best, optimal or near-optimal solutions to a given optimisation problem is called optimisation.

Each optimisation problem has one or more objective functions which should be minimised or maximized. These functions consist of one or more dependent variables to which integer or real values can be assigned, subject to a set of constraints.

Combinatorial optimisation problems are optimisation problems which are used to determine numerical values for certain discrete variables, namely those which can only take on a finite number of values with respect to a given objective function, to achieve an optimal solution [40]. The job scheduling problem in grid computing in this thesis is a combinatorial optimisation problem.

A combinatorial optimisation problem can be formulated as a model with triple entries (S, f, Ω), where [37] [38]:

- S is a search space which consists of a finite set of candidate solutions.
- $f : S \rightarrow R_0^+$ is the objective function to be minimised or maximised, which, for every solution $s \in S$, assigns a non-negative value $f(s) \in R_0^+$.

- Ω is a set of constraints. A solution s that satisfies all the constraints in Ω is said to be a feasible solution and the set of all feasible solutions S_Ω, $S_\Omega \subset S$, are those solutions in S which satisfy all the constraints in the set Ω.

- Assuming that f is a minimisation function, a solution s^*, $s^* \in S_\Omega$, is said to be a global optimal solution if it satisfies the following constraint:
$$f(s^*) \leq f(s) \; \forall s \in S_\Omega.$$
Let S^*_Ω be the set of all global optimal solutions, such that $S^*_\Omega \subseteq S_\Omega$.

- The goal is to find at least one global optimal feasible solution, s^* ($s^* \in S^*_\Omega$), that gives the best objective value in terms of the objective function.

In this thesis, the work focusses on minimization problems where the best solution is the one with the lowest value with respect to the objective function.

2.2 Heuristic, meta-heuristic and hybrid meta-heuristic algorithms

There are two main approaches by which to solve optimisation problems, namely exact algorithms and heuristic algorithms. The former approaches are those computational algorithms which have been developed in such a way that they provide a guarantee of obtaining the optimal solution within a reasonable timeframe. However, for some optimisation problems, such as NP-hard problems, the time to provide such a solution increases exponentially as the size of the problem instances increases.

Alternatively, heuristic algorithms are optimisation methods which provide approximate, near-optimal solutions to complex high-dimension optimisation problems within a reasonable timeframe. Heuristics were first introduced in the 1970, and since then they have been applied successfully to solve numerous complex optimisation problems. Their success is primarily the result of their interesting characteristics such as simplicity, speed, their reduced requirements for additional information about the optimisation problem under investigation, and robustness in terms of performance. Although they do not guarantee an optimal solution to the problem at hand, heuristic algorithms tend to return good quality solutions in a reasonable time. Heuristic algorithms are extremely reliant on experts' experience and knowledge, that is, they are highly specific and problem-dependent algorithms.

On the other hand, a meta-heuristic algorithm is a high-level problem-independent algorithmic framework which provides a set of guidelines or strategies to develop heuristics that are applicable to various optimisation problems. Since it guides heuristics over the problem state space, a meta-heuristic algorithm will have the ability to explore its best

parts, allowing high-quality solutions to be obtained. In fact, meta-heuristics can be defined as heuristics about heuristics, that is, a meta-heuristic is also a heuristic, but a more powerful one, as it provides a mechanism to escape from local minima. Similar to heuristic algorithms, meta-heuristic algorithms are characterized by the fact that when they are used to solve an optimisation problem, they do not require any special knowledge about it. Examples of meta-heuristics include: Tabu Search (TS), Variable Neighbourhood Search (VNS), Simulated Annealing (SA), Evolutionary Computation (EC), Differential Evolution (DE), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO) and Artificial Bee Colony Optimization (ABC), to name just a few.

Meta-heuristic approaches have all proven their effectiveness in terms of solving different complex optimisation problems. However, hybridizing two or more meta-heuristics shows better performance than applying a stand-alone approach [10]. The new high-level meta-heuristic will inherit the best features of the hybridized algorithms, increasing the chances of skipping away from a local minimum, and hence enhancing overall performance [170].

There are three important issues that must be examined in advance in order to design a new hybrid meta-heuristic algorithm [170], namely:

1. The number of algorithms to hybridize

   Typically, there are no constrains on the number of algorithms that should be hybridized. However, the literature shows that most of the proposed methods have used only two or three algorithms.

2. The type of algorithms to hybridize

   This aspect includes the selection of the methods to be hybridized from the available approaches in the domain, i.e., deterministic heuristic, meta-heuristic, local search, bio-inspired algorithms, etc.

3. The hybridization level

   This represents the most important aspect in the design of a new hybrid algorithm, and which refers to the degree of coupling between the selected methods, and the execution flow.

   (a) Degree of coupling

   This denotes the sequence in which the selected methods are executed. Two types of coupling can be defined, loosely coupled and strongly coupled, which can be described as follows:

   - **Loosely coupled:** This is also called a high level of hybridization. In this type of hybridization, the hybridized algorithms preserve their identity

through being run as a chain of executions in which the output of the first algorithm is used by the second, and so on; the final solution will be the output of the last algorithm. The execution of this chain can be repeated many times until stopping criteria are met. This kind of coupling can be denoted as Algorithm1 + Algorithm2, where Algorithm1 is first executed and its output is further improved by Algorithm2. Algorithm 2.1 shows a typical scheme for a loosely coupled hybrid meta-heuristic algorithm.

Algorithm 2.1: Schema of a loosely coupled hybrid meta-heuristic algorithm.

```
1  Function meta-heuristic algorithm-1(Initial-Solution)
2  begin
3  |   Processing;
4  |   return Solution-1;
5  end
6
7  Function meta-heuristic algorithm-2(Solution-1)
8  begin
9  |   Processing;
10 |   return Solution-2;
11 end
12     .
13     .
14     .
15 Function meta-heuristic algorithm-n(Solution-n-1)
16 begin
17 |   Processing;
18 |   return Final-Solution;
19 end
```

- **Strongly coupled:** This is also called a low level of hybridization. In this type of hybridization, the inner procedures of the hybridized algorithms are interchanged. In general, one of the methods acts as the main algorithm which, during its execution, calls other methods to act as supporting algorithms. For example, the genetic algorithm can be run and then a local search operator might be applied after performing the normal genetic operators. The local search then will help to improve the newly generated individuals. This kind of coupling can be denoted as Algorithm1(Algorithm2), where Algorithm1 represents the main algorithm and Algorithm2 represents the supporting algorithm. Algorithm 2.2 illustrates a typical scheme for a strongly coupled hybrid meta-heuristic algorithm.

Algorithm 2.2: Schema of a strongly coupled hybrid meta-heuristic algorithm.

```
1  Function main meta-heuristic algorithm()
2  begin
3      Generate(Initial-Solution);
4      Solution-1 ← Processing(Initial-Solution);
5      Call meta-heuristic algorithm-1(Solution-1);
6      return Solution-2;
7      Solution-3 ← Processing(Solution-2);
8      Call meta-heuristic algorithm-2(Solution-3);
9      return Solution-4;
10         .
11         .
12         .
13     Solution-n-1 ← Processing(Solution-n-2);
14     Call meta-heuristic algorithm-n(Solution-n-1);
15     return Final-Solution;
16 end
```

(b) Execution flow

The execution flow refers to the type of computing environment, namely whether this environment is sequential or parallel. In the sequential mode, hybridized algorithms are run sequentially (or in serial), while the parallel mode offers the opportunity to simultaneously run multiple algorithms (i.e., in parallel) using a network-based computing environment.

In order to provide a robust background to the work accomplished in this thesis, in the following sections we will provide detailed descriptions of the concepts behind three meta-heuristic algorithms (VNS, ACO and GA) that will be used to solve the job scheduling problem in grid computing.

2.3 Variable Neighbourhood Search (VNS)

VNS is a simple and effective meta-heuristic algorithm proposed by Mladenović and Hansen in 1997 [114]. It represents a flexible framework through which heuristics for solving a set of optimisation problems can be built. VNS uses multiple neighbourhood structures to explore the various neighbourhoods of the current incumbent solution, and then selects the one which results in an improvement. The underlying principle to VNS is based on making systematic changes to these structures both in the descent phase, in which the algorithm tries to find a local minimum, and the perturbation phase, in which VNS tries to escape from local minima.

Generally speaking, VNS consists of three steps, the shake step, the improvement step and the neighbourhood change step, as illustrated in Algorithm 2.3. These steps are repeated until some stopping conditions are met.

The goal of the shaking step is to resolve local minimum traps by applying a set of operators in a particular order. Each operator modifies a given solution S using predefined neighbourhood structures. The neighbourhood structure provides a means with which to explore new parts of the solution space. This exploration is achieved through defining the type of modifications which could be applied to a given solution to produce new solutions. The solution space can be explored in different ways using different neighbourhoods; thus, using well-defined neighbourhood structures will certainly lead to better exploration of the solution space.

The improvement step involves the application of a local search operator in an attempt to improve the solution produced by the shaking step. Two common search strategies are used within the improvement step, the first improvement and the best improvement. The former strategy stops the local search procedure as soon as an improvement to the current solution is encountered, while the latter checks all possible solutions obtained by local search and selects the best among them. Beside these strategies, it is also possible to use meta-heuristic algorithms for the local search.

The neighbourhood change step, which is the final step in the basic VNS procedure, is used to make a decision about the neighbourhood to be explored next and whether to accept the current solution as a new incumbent solution or not. Various neighbourhood change procedures are available in the literature, such as the sequential, cyclic and pipe neighbourhood change procedures [72].

Algorithm 2.3: The basic VNS procedure

```
1  Let: S_0 ←initial solution;
2  Let: N_k ←the set of neighbourhood structures, k ∈ [1, k_max];
3  repeat
4      k ← 1;
5      repeat
6          S_1 ← shake(S_0);
7          S_2 ← local_search(S_1);
8          if (fitness(S_2) < fitness(S_0)) then
9              S_0 ← S_2;
10             k ← 1;
11         else
12             k ← k + 1;
13         end
14     until (k==k_max);
15 until (termination condition);
```

2.4 Ant Colony Optimization (ACO)

The appearance of ants on Earth goes back some 100 million years. Currently, the estimated ant population is about 10^{16} individuals [76]. Ants are social insects which live in colonies, the size of which varies from just 30 to millions of ants. The complex behaviour of ants in these colonies has long encouraged entomologists to further investigate this collective behaviour, which has resulted in an improved understanding of their working mechanisms. Foraging for food, building nests and division of labour are just some of the examples of the complex collective behaviours which have been intensively explored. The emergence of such is due to the collective behaviour of very simple social insects which act as stimulus–response agents. Each agent can interact with its environment, i.e., receive communications, and as a response can carry out basic, unsophisticated actions with a large random component [45].

The collective behaviour of ant colonies was first investigated in 1927 by Eugene Marais who published his findings in his book "The Soul of the Ant" [104]. After performing several experiments, the author provided a full description of how ant colonies work. These findings were used by Maurice Maeterlinck, who wrote "The Life of the White Ant" [97], to provide a further description of ants' complex behaviour.

From these primary studies, the means by which ants communicate was hypothesized by Pierre-Paul Grasse [69] when he explored ants' behaviour as they construct their colonies. In his findings, he noticed that ants interact indirectly, for which he used the term 'stigmergy' to describe this phenomenon. Furthermore, he concluded that the complex collective behaviour of ant colonies is a result of this type of indirect interaction, or communication, which occurs between the ants themselves and between ants and the environment. On the other hand, another type of stigmergy, which is called pheromonal stigmergy, was examined by Deneubourg et al. [34]. Consequently, these works paved the path to develop and implement the first algorithmic model to simulate the foraging behaviour of ants [35].

Ant Colony Optimization (ACO) is a meta-heuristic search algorithm that mimics the behaviour of ants when searching for a path between their nest and a source of food [38]. Despite the fact that ants are blind, they can search complex environments that are quite remote from their nest in an attempt to find food sources and to successfully carry that food back to their nest. This is achieved through laying a substance called *pheromone* on the ground while they move back and forth from their nest to a source of food. This substance can be sensed by other ants and allows indirect communication among them. This process, *stigmergy*, allows the ants to modify their environment to further allow the interaction between them and the colony and to remember the return path to their nest.

Ants which find the shortest path to the source of food will, then, return back to their colony earlier than ants following longer paths, which means that the shortest path has

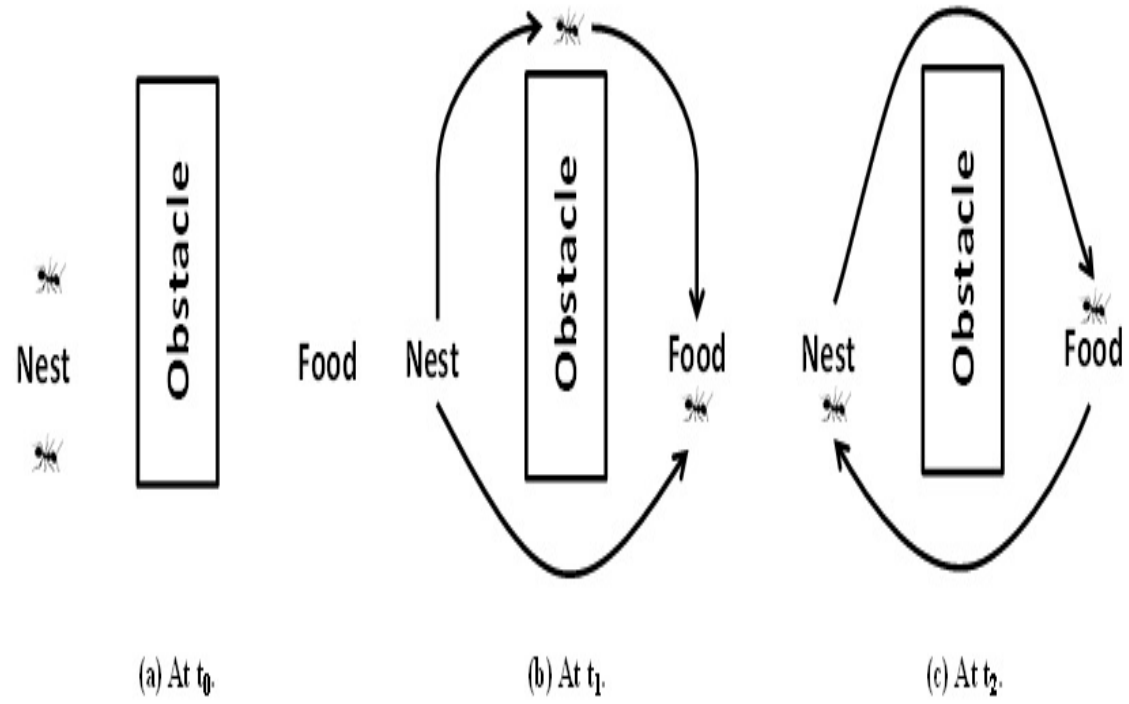

Fig. 2.1 Ant example.

been marched over more than other paths and thus that its pheromone density will be higher. Ants will choose the path with the highest pheromone concentration, i.e., the path with the highest pheromone levels will attract more ants building up even more pheromone. However, if this path remains after the consumption of the food it led to, this would seriously obstruct the ants' ability to find further food. To cope with this situation, pheromone trails evaporate over time, which is a mechanism by which to forget old decisions [41].

Consider the example given in Fig. 2.1, which illustrates how ant-based meta-heuristic algorithms allow for path optimisation using the principle of indirect communication between autonomous agents. In Fig. 2.1(a), two ants at their nest at time t_0 are trying to reach the food source which exists on the opposite side of an obstacle. While navigating, a small amount of pheromone is deposited by each ant along the path as a signpost by which to return to their nest.

Each of these ants will choose a different path with equal probability, e.g., the upper path will be selected by the first ant and the lower path will be taken by the second ant. Let us assume that the upper path is twice the distance to the food source than the lower. Therefore, the second ant, which selected the lower path, will reach the food source at time t_1, while the first ant which took the upper path will reach only the half distance to the food source in the same time, as shown in Fig. 2.1(b).

The lower ant will reach the food source quicker than the upper ant. It then fetches some food and carries it back to the nest. On its way back, it will follow the same route, the shortest route, since no other signs are available to it. At time t_2, some food will be carried to the nest by the lower ant which reached its source successfully. During its journey back to the nest, the lower ant also lays down more pheromone which means the shortest path will contain higher concentrations of pheromone. On the other hand, the upper ant will finally arrive at its destination during the same time, as shown in Fig. 2.1(c).

Consequently, during the time interval $[t_1, t_2]$, the upper ant would have deposited pheromone on the upper path once while the lower ant has laid down pheromone on the lower path twice. This means that the pheromone levels on the lower path, the shortest path, are twice that of the upper path, which reinforces the shortest path. At the following time, based on the available pheromone levels, the upper ant will be more likely to follow the lower path.

2.4.1 The Simple Ant Colony Optimization

ACO-based meta-heuristic algorithms use a colony, or a population, of simple artificial ants (agents) which iteratively build candidate solutions to an optimisation problem. Ants build these solutions using two types of information, namely pheromone trails and heuristic information. The first step in any ACO-based algorithm is to determine what information the pheromone trail encodes. The pheromone trail, τ, allows the ants to communicate indirectly to each other and share useful information about optimal solutions. In addition to the information encoded in the pheromone trial, the ants use heuristic information, η, to build their solutions; this information is problem dependent.

Ideally, ACO-based meta-heuristic algorithms can be used to solve any optimisation problem by selecting an efficient way to represent the problem which clearly defines each component of the solution. The ants will use this representation to iteratively build candidate solutions through which pheromones are updated.

Algorithm 2.4: Simple ACO meta-heuristic algorithm.

```
1 Parameters setting and pheromone trails initialization;
2 while (stopping condition is not true) do
3 |   Build-Solutions();
4 |   Perform-Local-Search();        /* optional */
5 |   Update-Pheromone-Trails();
6 end
```

Generally speaking, to solve an optimisation problem using an ACO-based meta-heuristic algorithm, the steps illustrated in Algorithm 2.4 are applied. ACO starts by setting values to a number of parameters and initializing the pheromone trial to some – usually

small – value. It then enters a loop that consists of three procedures, and which will be repeated until a stopping condition, such as a maximum number of iterations, is met. The three procedures are: Build-Solutions(), Perform-Local-Search() and Update-Pheromone-Trails().

In the first procedure, Build-Solutions(), feasible solutions to the problem are constructed by the ants. Each ant starts with an empty solution. Components are iteratively added to update the solution until a complete feasible solution is constructed. The addition of a component to the solution is decided according to a probabilistic rule that each ant applies when they use the available heuristic and pheromone information. This point is called a choice point and can by defined by Equation 2.1 as follows:

$$p_{xy} = \frac{[\tau_{xy}]^{\alpha} * [\eta_{xy}]^{\beta}}{\sum [\tau_{xy}]^{\alpha} * [\eta_{xy}]^{\beta}} \tag{2.1}$$

where x and y are components of the solution, and α and β are parameters used to define the relative weights of the pheromone and the heuristic, respectively.

The second procedure, Perform-Local-Search(), is optional, and which might be used to further improve the solutions generated from calling the first procedure by performing a local search algorithm. The solutions achieved by ants may not be optimal, so one way in which to improve these solutions is to hybridize ACO with local search techniques, i.e., to apply local search techniques on the solution obtained by an ant in order to improve it further. The literature shows that for several static NP-hard combinatorial optimisation problems, the best reported results are achieved when hybrid ACO algorithms are used [21] [36] [61] [103] [143] [144] [145] [146].

Finally, in the Update-Pheromone-Trails() procedure, feedback to the constructed solutions is given by updating the pheromone levels of the components used to generate the solutions. These feedbacks allow the indirect communication among ants regarding the current status of the solution components. Normally, those components which have been used by many ants or have been included in the best solutions achieved at a given point, will get more pheromone than others. As a result, the chances that the ants will use them during subsequent iterations are higher than other components. The pheromone trial can be updated using a simple rule, as defined by Equation 2.2:

$$\tau_{xy} = \rho * \tau_{xy} + \Delta\tau_{xy} \tag{2.2}$$

where x, y are the solution components that belong to the best solution, ρ $(0 < \rho \leq 1)$ is the decay parameter which is used to allow the ants to forget poor information, i.e., to avoid sticking in a local minimum, and $\Delta\tau_{xy}$ is the amount of pheromone deposited between the solution components.

2.5 Genetic algorithm (GA)

A GA is a meta-heuristic search algorithm which was developed at the University of Michigan by John Holland in 1975, and which is based on biological evolution, i.e., it uses operations that exist in nature [130] [136]. It combines a randomized information exchange structure with the mechanism of natural selection (Darwinian survival of the fittest concept) [39]. GAs can rapidly find near optimal solutions by searching large and complex state spaces in an efficient manner [130].

A GA undergoes an evolutionary process to solve a problem, i.e., the solutions are evolved. It works on a group of solutions (individuals or chromosomes), called the population, rather than on one solution only. It first generates an initial population randomly. Each solution in the population is evaluated using a fitness function which assigns a score that indicates the solution's quality. The GA then evolves toward an optimal solution after a number of generations through applying the genetic operators, namely selection, crossover and mutation.

To solve a problem P using a GA, the following steps, which are illustrated in Fig. 2.2, are applied [116]:

1. Select a proper representation to encode P, determine k, which is the number of individuals (the size of a population), the crossover probability (pc) and the mutation probability (pm).

2. Define a proper fitness function, which depends on P's domain. This function is used to evaluate each solution (individual) in the population by assigning a score to each one that represents the quality of that solution in the problem domain. The algorithm later uses these scores to select individuals for mating.

3. Generate an initial population, usually randomly, of individuals of size k:
$$S_0, S_1, ..., S_{k-1}.$$

4. Evaluate the fitness of each individual:
$$fintness(S_0), fitness(S_1), ..., fitness(S_{k-1}).$$

5. Use the current population to probabilistically select two parent individuals for mating according to their fitness. Individuals with high fitness scores will have a higher chance to be selected for mating than others.

6. Apply the genetic operators (crossover and mutation) to produce two new offspring.

7. Add the two new offspring to the new population.

8. If the size of the new population $<$ k, go to Step 5.

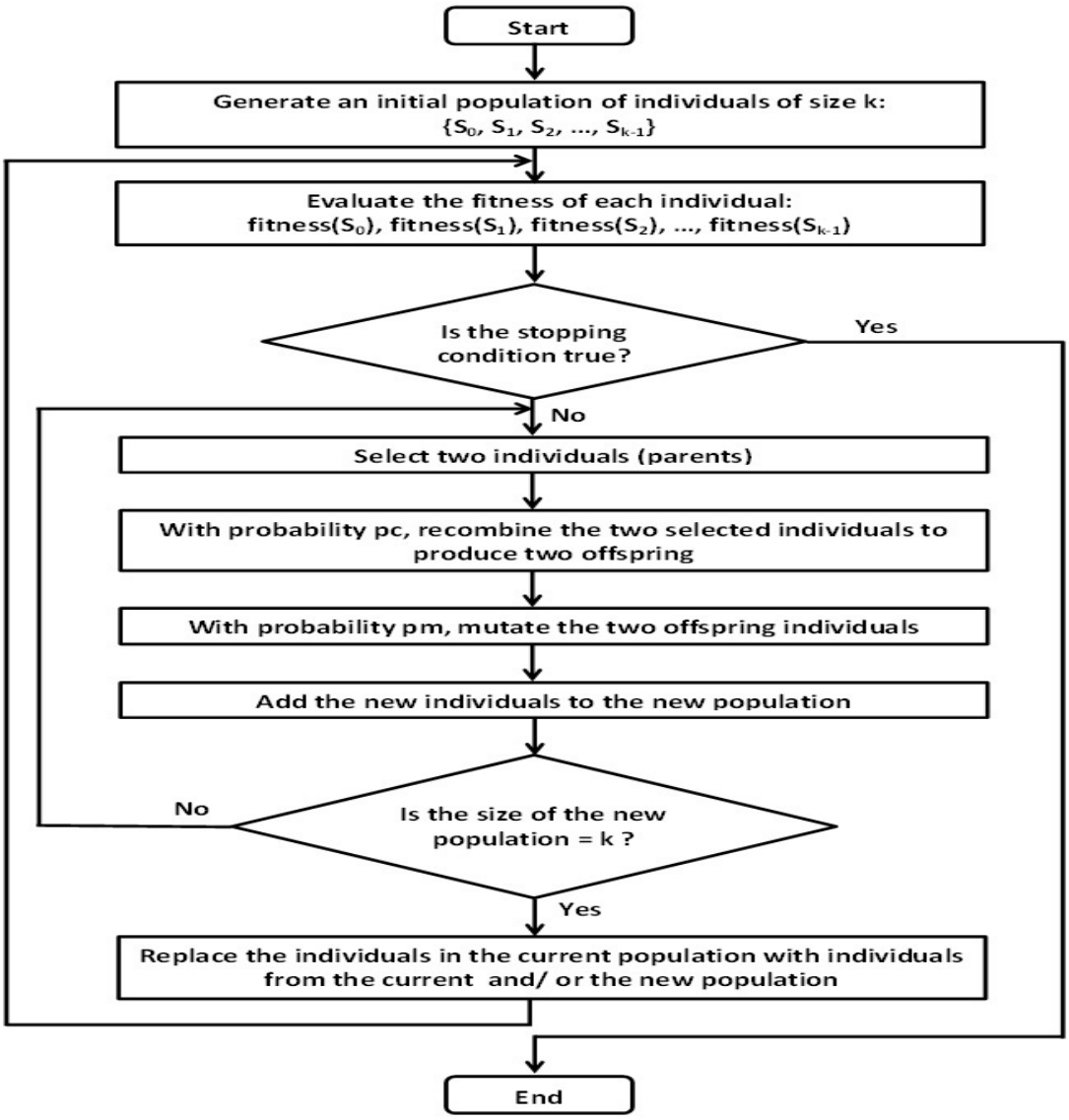

Fig. 2.2 A flowchart for a genetic algorithm.

9. Replace the parent population with the new offspring population.

10. If the stopping condition has not been met, go to Step 4 and repeat the process.

As the above steps state, a GA is an iterative process. Each iteration represents a generation and the entire set of generations represents a run. One (or more) highly fit individual(s) is expected to be found at the end of a run.

The main issues that need to be considered in GAs are:

- How to encode the solutions?

- How to create an initial population?

- How to evaluate each solution in the population?

- Which solutions should be selected to perform mating?
- How to alter the selected solutions to generate new, hopefully better, offspring.
- Which solutions should be selected to survive for the next generation?

The answers to the above questions represent the main ingredients of a GA. Each of these ingredients will be examined in turn in the following subsections.

2.5.1 Encoding

How to encode or represent solutions is the first issue that needs to be considered when a GA is selected for application to a problem. Encoding relies heavily on the problem. The literature on GAs includes several encoding schemes. One of the most common and simple encoding methods is that of binary encoding, in which individuals are represented as a sequence of bits (zero or one). Although simple, binary encoding is often not a natural choice for a wide range of problems and corrections might be required after performing certain genetic operations.

Another example of an encoding method is permutation encoding, which is usually used in ordering problems. In this encoding, an individual is represented as a string of numbers, each of which indicates a unique position in a sequence. Unlike the binary encoding, repetition is not allowed. One example in which permutation encoding might be used is the Travelling Salesman Problem (TSP). In a TSP, N cities and the distances between them are given. The travelling salesman has to start from one city then visit all the remaining cities, and finally return to the starting city without visiting any city more than once (except the starting city). The goal is to find a sequence of cities that satisfies the above conditions and best minimises the total distance travelled (the summation of the distances between the visited cities). One way to encode the solutions to this problem is permutation encoding, in which it is assumed that each individual describes the order of cities in which the salesman will visit them. For example, let N = 5, the individual S = 3 2 5 1 4 means that the salesman started from city 3 then visited city 2 and so on until he arrived at city 4.

Permutation encoding is useful for ordering problems such as in the task ordering problem, in which each number should be unique (repetition is not allowed). However, proper types of crossover and mutation operators should be used to guarantee that the new offspring have no repetitions (for instance in TSPs, the same city should not appear twice).

2.5.2 Initialization

After selecting the encoding scheme that properly represents the problem, the next step is to determine the size of the population (the number of individuals). This size is not fixed, and thus differs from one problem to another; hence, it should be tuned.

After selecting the size of the population, an initialize population can be generated that contains diverse individuals. Generally speaking, initialization is considered to be one of the main considerations in GAs. Various studies have discussed the effects of generating a bad initial population (suffers from lack of diversity), and they came to the conclusions that this prevents GAs from converging towards the global optimum and increases the time required to find good solutions [82]. The diversity in the population is crucial as individuals must learn from each other and, more importantly, to avoid premature convergence to sub-optimal solutions, which is a problem GAs can suffer from when there is a lack of diversity in the population [95]. There are various methods by which to maintain diversity in the initial population which include:

1. Uniformly random

   A uniform distribution is used to generate individuals randomly from the search space of the problem.

2. Grid initialization

   In this method, certain parts in the search space are selected to seed the initial population. The selection of these parts is generally problem dependent.

3. Non-clustering rule

   Another way to ensure diversity is to place a restriction on the newly generated solutions. A rule is defined to make sure that when a new individual is generated it must differ from all previously added individuals in the population by a predefined number of genes.

4. Heuristic

   Deterministic *ad-hoc* heuristics, such as hill climbing and best-first search, and other search techniques can be used to generate the initial population in GA. This is motivated by the fact that a GA will obtain a solution which is at least as good as the seeded solution.

2.5.3 Fitness function

Each individual in the population is evaluated using a fitness function which assigns a score to indicate the solution's quality. Later on, these scores are used to select which individuals

can be reproduced to provide, hopefully, better offspring. Consequently, to design a good fitness function that can easily differentiate between good and bad individuals, careful consideration is required.

2.5.4 Selection

The selection operator refers to the process that determines which individuals are to be continued and allowed to reproduce and which ones deserve to be eliminated. This selection depends heavily on the fitness function, several selection techniques are available in the literature. However, choosing the right selection operator, namely one which avoids picking weak solutions and allows for the survival of good solutions to the problem under discussion, is problem dependent. Some examples of common selection techniques include:

1. Random selection

   In this type of selection, which is considered the simplest selection operator, individuals are selected randomly regardless of their fitness, which gives all individuals (best and worst) an even chance of being selected. Compared to other selection operators, random selection has the lowest selective pressure towards fitter individuals.

2. Proportional selection

   This is one of the most popular selection operators which was proposed by Holland [75]. Unlike the random selection operator, proportional selection is biased towards fitter individuals, i.e., it chooses individuals according to their fitness. Each individual has a probability of being selected which is proportional to its fitness. Therefore, the best individuals in terms of fitness score are the more likely to be selected for reproduction. Consequently, better individuals are expected to evolve over time as a high selection pressure towards fitter individuals in the population in such a selection operator is applied.

   Proportional selection can be implemented in two ways, namely roulette wheel selection and stochastic universal selection.

   The former implementation normalizes individuals' fitness scores by dividing each score by the maximum fitness score. Table 2.1 shows an example of a hypothetical population of six individuals along with their fitness score and their corresponding normalized score. The probability distribution can then be represented as a circular wheel (roulette wheel) that contains k pies, where k is the number of individuals in the population. Each individual occupies a slice of the roulette wheel that is proportional to its normalized fitness score. Selection can be made by rotating the roulette wheel and using the individual whose portion in the roulette wheel comes

Table 2.1 Hypothetical Population

Individual	Fitness score	Normalized fitness
A	14	0.27
B	9	0.17
C	5	0.10
D	6	0.12
E	11	0.21
F	7	0.13
sum = 52		

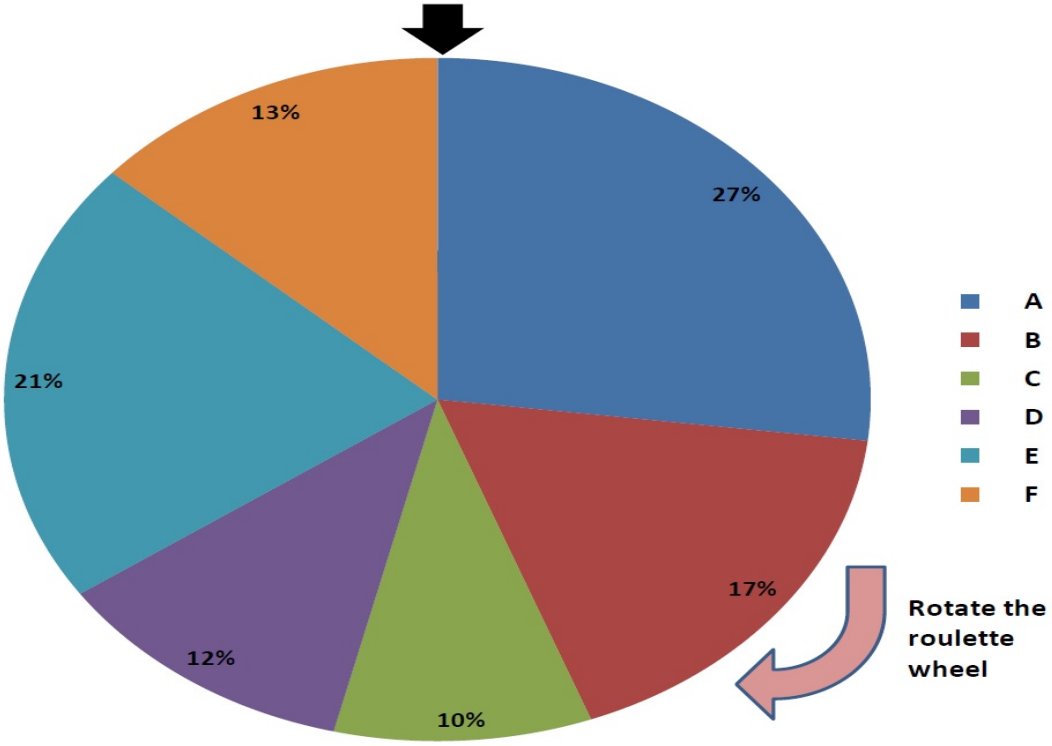

Fig. 2.3 Example of Roulette wheel selection.

at a previously determined fixed point is selected. Fig. 2.3 shows how the roulette wheel selection can be applied to the hypothetical population of Table 2.1, where the fixed point is located at the top of the figure.

The latter implementation follows the same steps as the roulette wheel selection, with the exception that it contains multiple fixed points instead of just one, as shown in Fig. 2.4. Therefore, more than one individual can be selected in just one rotation of the wheel, increasing the probability of choosing the more highly fit individuals at least once.

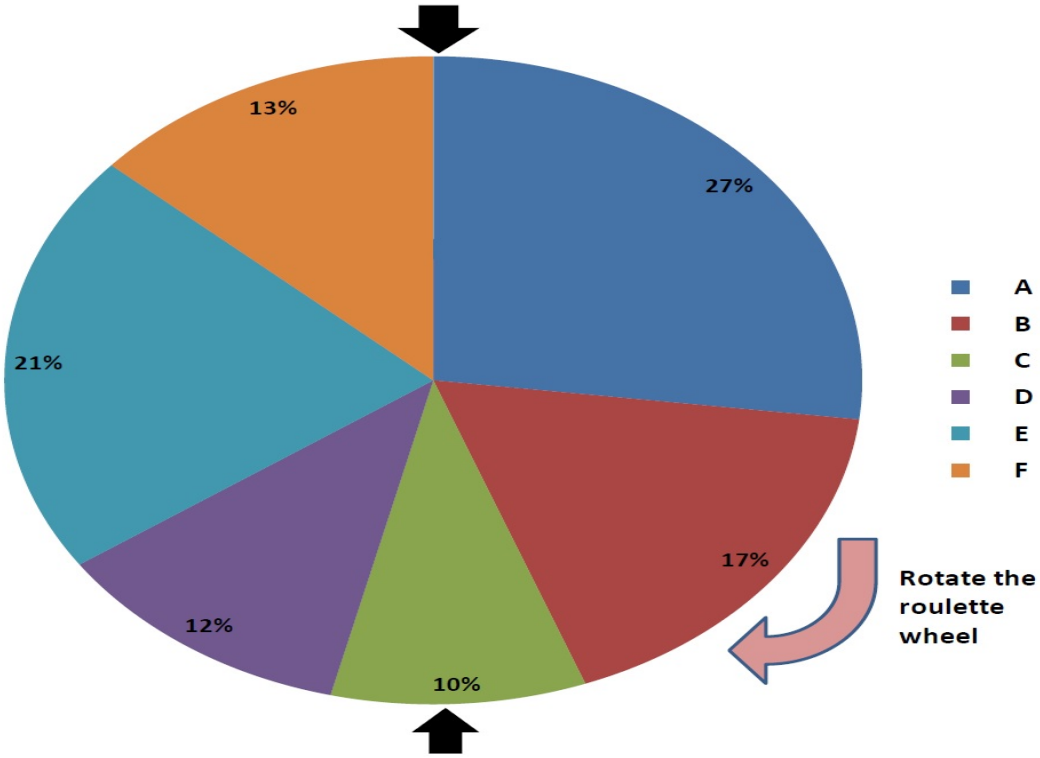

Fig. 2.4 Example of Stochastic Universal Sampling.

3. Tournament selection

   In a tournament selection, several tournaments are run among a few individuals who have been randomly selected from the population. The winner of each tournament (the one with the best fitness) is selected for subsequent stages.

2.5.5 Alteration

This involves applying a number of genetic operators that are used to modify the individuals in the population. Two common genetic operators can be used to alter individuals, which are as follows:

2.5.5.1 The crossover operator

The crossover operator is equivalent to reproduction and biological crossover. New solutions (offspring) are generated by selecting individuals from the parental generation and exchanging their genes. Crossover enables the search process to explore new regions of the solution space and provide the next generation with good quality individuals.

Several types of crossover operators have been reported in the evolutionary computing literature, which mainly depend on the solution representation. Therefore, in our case, three crossover operators, which are the one-point (1P), two-point (2P) and Half Uniform

Crossover (HUX), will be considered for the direct representation, while another three operators will be examined for the permutation-based representation, which include Order Crossover (OC), Partially Matched Crossover (PMC) and Cycle Crossover (CX).

1. One-Point Crossover (1P)

   1P represents the pioneer crossover operator, which was proposed in reference [75]. Given two parent solutions, the one-point crossover operator starts by generating a random position between 1 and the individual's size-1. This position serves as an exchange point which divides each parent into two parts. Two new offspring are obtained by exchanging the two first segments of the parents. Fig. 2.5 shows an example of 1P crossover.

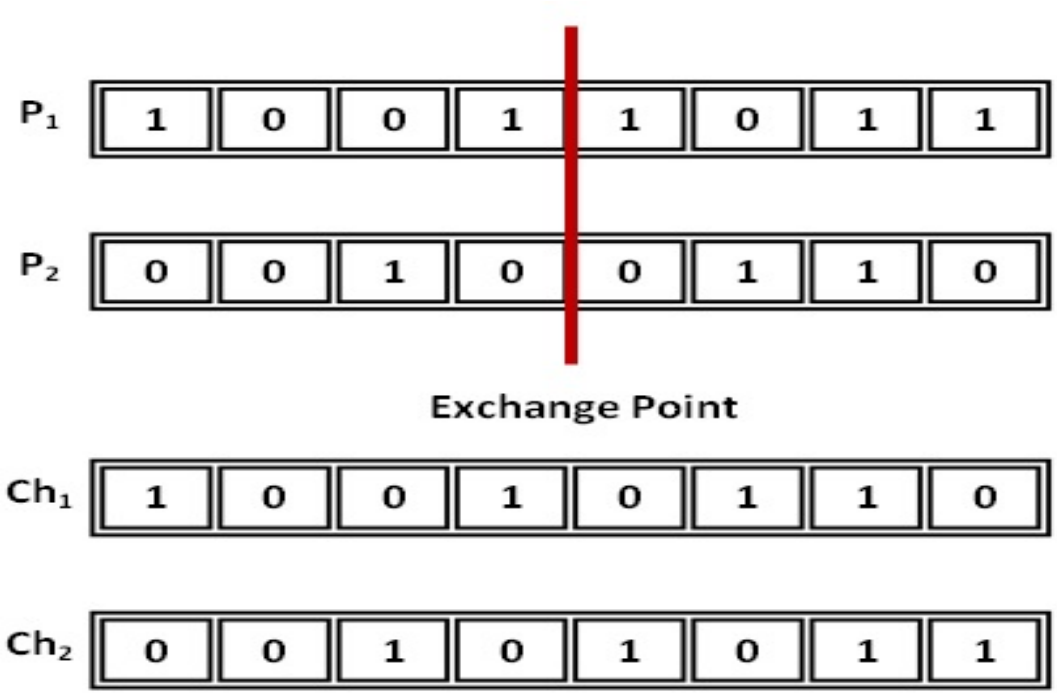

Fig. 2.5 Example of One-Point Crossover.

2. Two-Point Crossover (2P)

   2P is the generalized form of the 1P crossover operator, as proposed in reference [33]. Given two parent solutions, unlike the one-point crossover, this operator starts by generating two random cutting points between 1 and the individual's size-1. These positions serve as exchange points which divide each parent into three parts. Two new offspring are obtained by exchanging the parental segments between the two cutting points. Fig. 2.6 shows an example of 2P crossover.

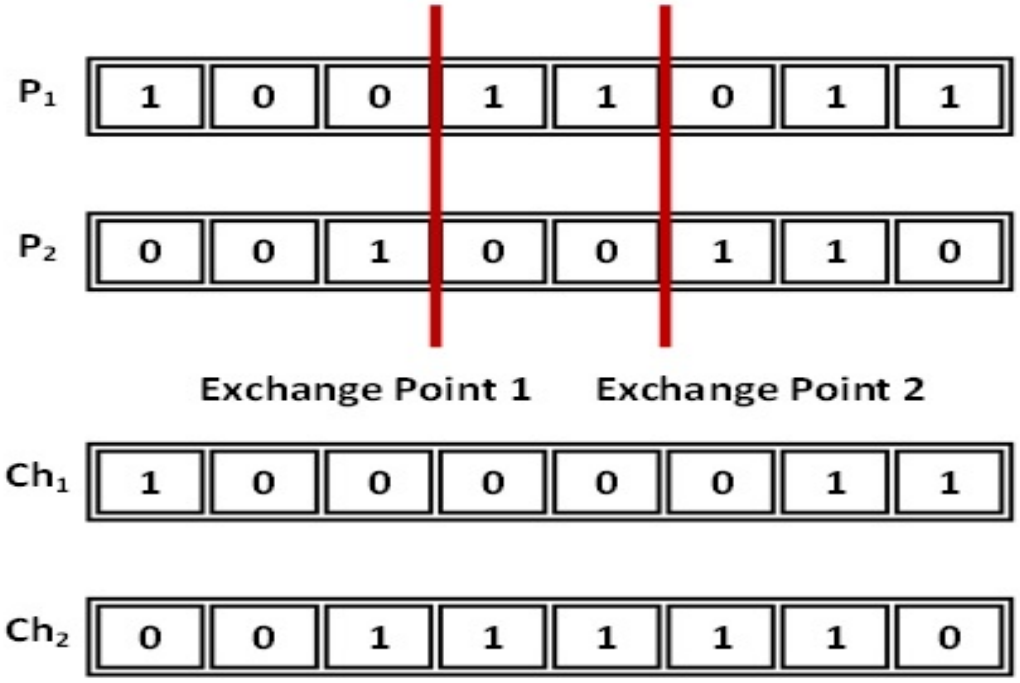

Fig. 2.6 Example of Two-Point Crossover.

3. Half Uniform Crossover(HUX)

   HUX, as first proposed in reference [150], requires that half of the non-matching genes are swapped; to achieve this, the Hamming distance – that is, the number of differing genes – is determined and subsequently halved, The calculation gives the number of bits that are unmatched genes between the two parents, and hence the number of genes that need to be swapped. The steps describing the HUX crossover operator are given in Algorithm 2.5. Fig. 2.7 shows an example of HUX crossover.

4. Order Crossover (OX)

   The Order Crossover is simple permutation crossover operator originally proposed by reference [32]. Fig. 2.8 shows an example of OX crossover. Basically, OX receives two parents, P_1 and P_2, and produces two offspring, Ch_1 and Ch_2, as follows:

   (a) Two crossover points, POS1 and POS2, are randomly selected and the block of genes between them from P_1 is copied to Ch_1.

   (b) Delete all genes in P_2 that are already in the copied block. The remaining genes are those that the offspring needs.

   (c) To create Ch_1, copy the remaining genes according to the order they appear and from left to right.

Algorithm 2.5: The HUX crossover algorithm

input : two parents P_1 and P_2 and their size n (number of genes);
output : two offspring Ch_1 and Ch_2;

$Ch_1 \leftarrow P_1$;
$Ch_2 \leftarrow P_2$;
$no\_of\_diff\_genes \leftarrow 0$;
for *(i = 1 to n)* **do**
  if $(Ch_1[i] \neq Ch_2[i])$ **then**
    $no\_of\_diff\_genes \leftarrow no\_of\_diff\_genes + 1$;
  end
end
$exchange\_counter \leftarrow 0$;
while $(exchange\_counter < (no\_of\_diff\_genes/2))$ **do**
  for *(i = 1 to n)* **do**
    if $((Ch_1[i] \neq Ch_2[i])$ *and* $(Ch_1[i] \neq P_2[i]))$ **then**
      generate a random number R;
      if $(R \leq 0.5)$ **then**
        $Ch_1[i] \leftarrow P_2[i]$;
        $Ch_2[i] \leftarrow P_1[i]$;
        $exchange\_counter \leftarrow exchange\_counter + 1$;
        if $(exchange\_counter < (no\_of\_diff\_genes/2))$ **then**
          Break;
        end
      end
    end
  end
end

Fig. 2.7 Example of Half Uniform Crossover.

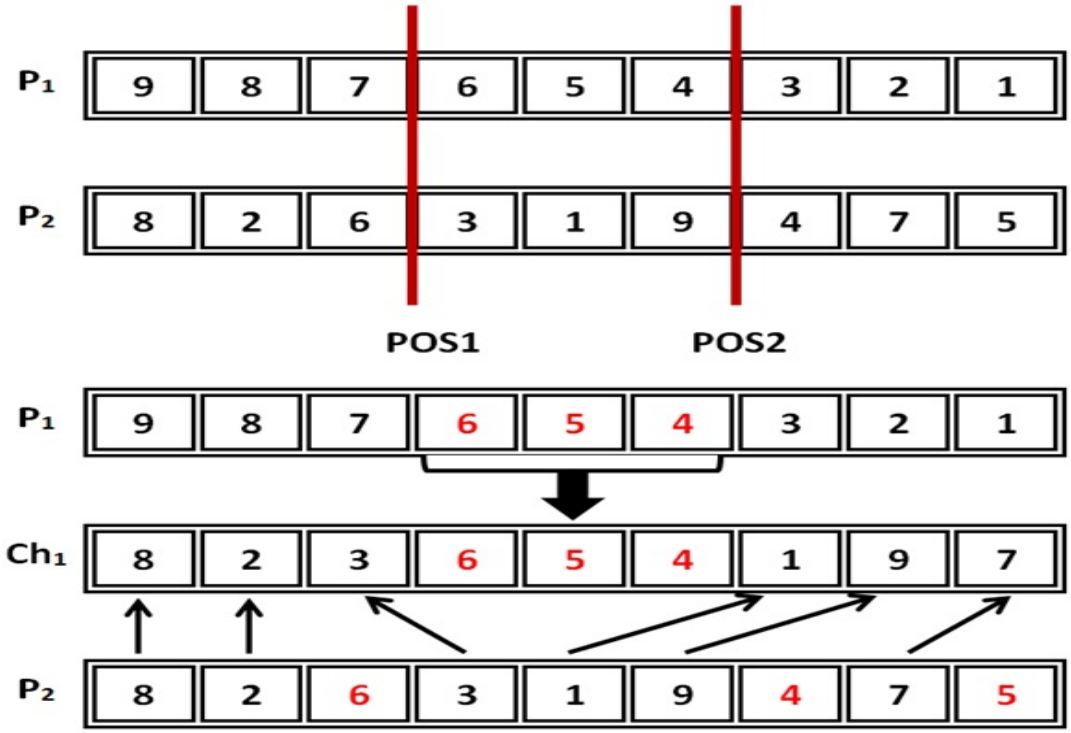

Fig. 2.8 Example of Order Crossover.

(d) Create Ch_2 by flipping P_1 and P_2 and repeating steps 1, 2 and 3.

5. Partially Matched Crossover (PMX)

PMX, also known as Mapped Crossover, is the mostly commonly used crossover operator for problems that use a permutation-based representation, as proposed in reference [66]. Fig. 2.9 shows an example of PMX crossover. Basically, PMX receives two parents, P_1 and P_2, and produces two offspring, Ch_1 and Ch_2, as follows:

(a) Two crossover points, POS1 and POS2, are randomly selected and the block of genes between them from P_1 is copied to Ch_1.

(b) Check the same positions in P_2 to find the values, V_i, that have not been copied to Ch_1.

(c) For every $v \in V_i$:

i. Find the corresponding value, v_0, in P_1 at the same position of v.

ii. Check v_0 in P_2; if it is part of the copied block then go to Step i and repeat using v_0; otherwise, insert v into Ch_1 in this position.

(d) Copy any remaining values from P_2 to Ch_1.

(e) Create Ch_2 by flipping P_1 and P_2 and repeating steps a, b, c and d.

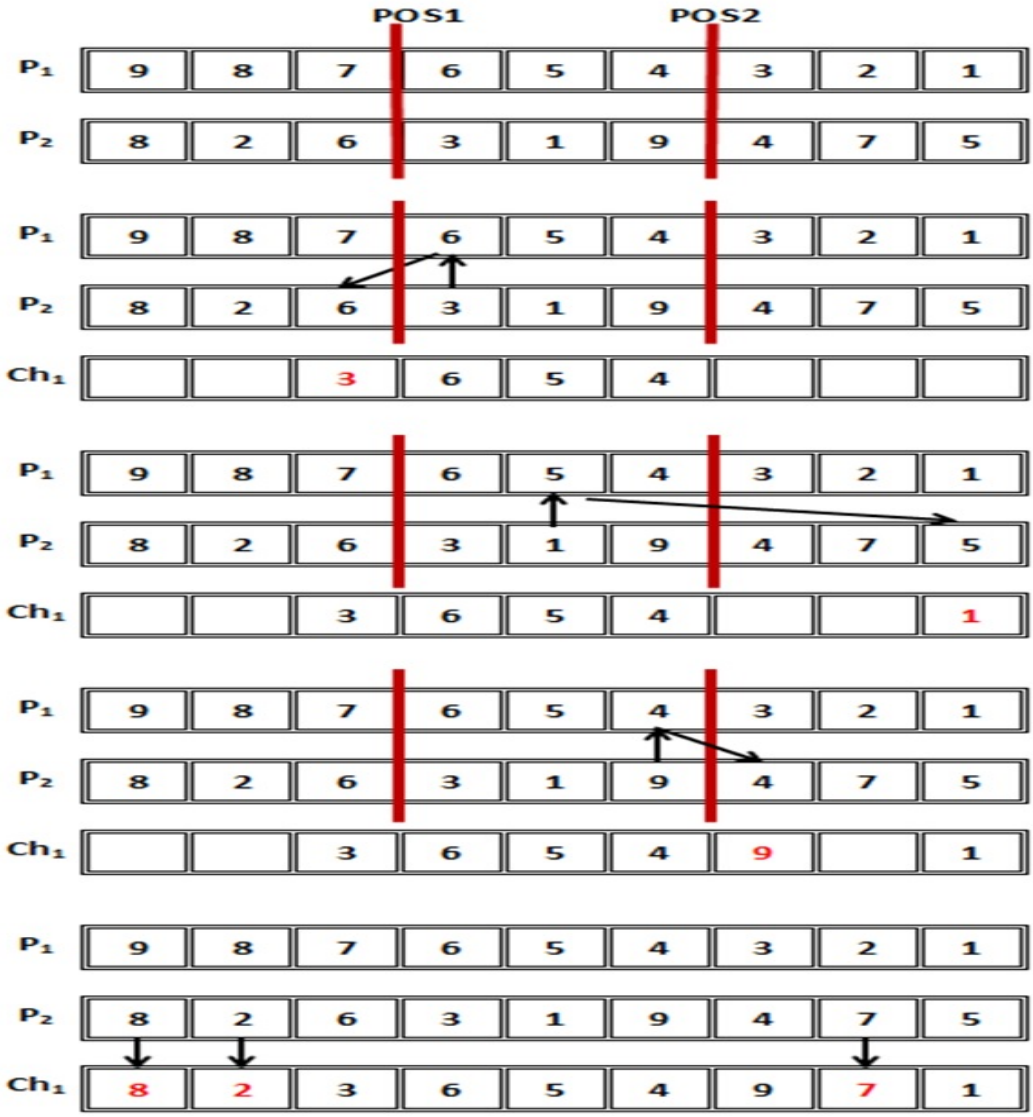

Fig. 2.9 Example of Partially Matched Crossover.

6. Cycle Crossover (CX)

   CX is another crossover operator which is used for permutation-based representation, as proposed by reference [120]. Fig. 2.10 shows an example of CX crossover. Basically, CX receives two parents, P_1 and P_2, and produces two offspring, Ch_1 and Ch_2, as follows:

   (a) Identify a number of cycles between the two parents. A cycle is a subset of genes that is constructed by starting from one of the parent genes and then going to the corresponding gene in the other parent, and so on until we reach the same gene that we started with.

(b) To generate Ch_1 and Ch_2, the first cycle is copied from P_1 to Ch_1 and its corresponding cycle is copied from P_2 to Ch_1, whilst the second cycle is copied from P_2 to Ch_1 and its corresponding cycle is copied from P_1 to Ch_2, and so on.

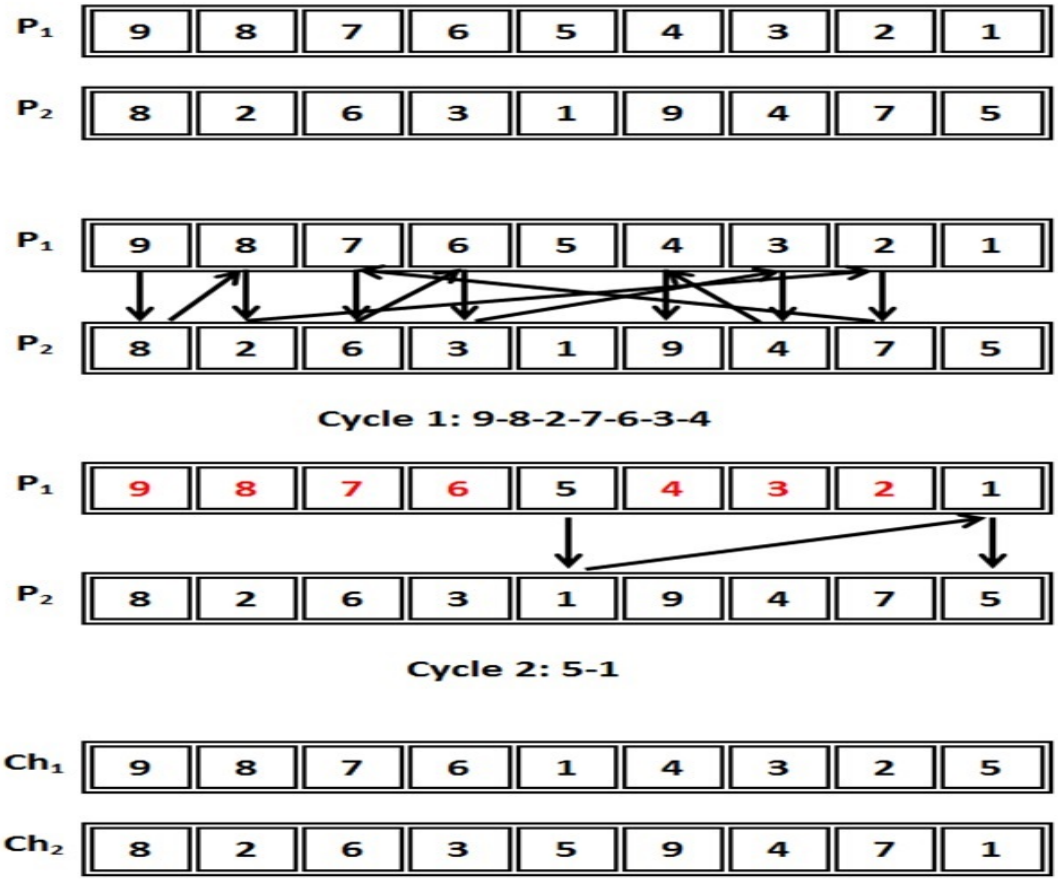

Fig. 2.10 Example of Cycle Crossover.

2.5.5.2 The mutation operator

The mutation operator is one of the most important elements of a GA, and which is related to the exploration of the search space. Through mutation, individuals are randomly altered to maintain and introduce diversity into subsequent generations [121]. The literature on GAs describes several mutation operators such as:

1. Random mutation

   The earliest mutation operator was proposed by reference [33]. In this type of mutation, a random position in an individual is selected and its value is flipped. This technique works perfectly for those problems where the binary representation can be used to encode them as it is easy to flip from 0 to 1 and vice versa.

Fig. 2.11 An example of the insert mutation.

2. Insert mutation

   In this operator, two positions, POS1 and POS2, such that 1 < POS1 < POS2 < n, where n is the length of encoding, are randomly selected from an individual, after which POS2 is inserted directly before POS1 and all positions from POS1 to POS2 are right shifted. Fig. 2.11 shows an example of the insert mutation.

3. Swap mutation

   Similar to the insert mutation operator, two positions, 1 < POS1 < POS2 < n, where n is the length of encoding, are randomly selected from an individual. However, instead of inserting POS1 before POS2, the swap operator merely exchanges their values. Fig. 2.12 shows an example of the swap mutation.

2.5.6 Replacement

Another operator that is also related directly to the Darwinian concept of survival of the fittest is the replacement operator; this is quite similar to the selection operator, but occurs at the end of each generation. It involves deciding which individuals are selected to survive into the next generation. The candidate individuals can be chosen from the offspring and/or parents. Similar to the selection operator, the main goal of the replacement operator is

Fig. 2.12 An example of the swap mutation.

to guarantee that fitter individuals will propagate to subsequent generations. Two main replacement operators are available which include:

1. Generational replacement

   In this type of replacement, all individual parents are replaced with their offspring, which means that there is no overlap between the current generation and the next.

2. Steady-State Strategy

   In this type, parents and offspring compete for survival, and then the best of them are selected to go on to the next generation. Therefore, there is overlap between the current and next generations.

2.6 Summary

This chapter has introduced the necessary theoretical background to optimisation problems and has discussed the various methods available in the literature to solve them. Among these methods, the meta-heuristic algorithms are considered the most suitable candidates as they are able to search large-size state spaces and provide high-quality solutions to a large and diverse range of complex optimisation problems. The chapter described what is meant by an optimisation problem, combinatorial optimisation problems, heuristics,

meta-heuristics and hybrid meta-heuristics. The chapter also described the main concepts behind a number of selected meta-heuristic algorithms which will be applied to solve the problem of job scheduling in grid computing. These algorithms include VNS, ACO and GA. VNS is a simple and effective meta-heuristic algorithm which represents a flexible framework for building the heuristics to solve a set of optimisation problems. VNS is based on the systematic change of these structures both in the descent phase, in which the algorithm tries to find a local minimum, and the perturbation phase, in which VNS tries to escape from the local minimum. ACO is a bio-inspired meta-heuristic search algorithm that mimics the behaviour of ants in searching for a path between their nest and a source of food. A GA, by contrast, is another bio-inspired meta-heuristic search algorithm that simulates the natural selection process of biological evolution.

Chapter 3

Job Scheduling in Grid Computing

This chapter introduces an overview of grid computing and its basic components. Additionally, the chapter presents the job scheduling problem description and the simulation model that mimics the assignment of jobs to resources. Furthermore, the main methods described in the literature to generate problem instances are also explained. Finally, it presents a comprehensive review of the static and dynamic heuristic and meta-heuristic approaches that were used to tackle the job scheduling problem in heterogeneous environments such as grid computing.

3.1 Grid computing

The advance in computer software, hardware and networking has led to a significant increase in commodity cluster computing in the last two decades. This type of computing has been used to provide powerful computing resources at a low-price cost to solve complex problems from various application domains which require an intensive use of resources. Specifically, the scientific society utilizes from these powerful computing resources which allow researchers and scientists to extend and run more experiments and simulations and to test more parameters. Results to these experiments, simulations and parameters tuning processes can be instantly shared among different geographically distributed partners due to the substantial increase in the communication bandwidth. This motivates universities and other education enterprises to start launching special programs, known as e-Science [74], to establish such cooperation to solve many complex large-size scientific problems. As a result of the intensive use of these cooperation by multiple researchers and scientists, massive distributed data were generated within e-Science programs. Thus, the main challenge that has faced e-Science programs was how to maintain data management, in this environment. This challenge, therefore, motivates the design of a distributed computational infrastructure which couples various resources such as storage spaces, databases, servers,

fast networks, clusters and supercomputers for solving a wide range of complex problems, emerging into the so-called Grid computing [15] [23] [56] [127].

Grid Computing has been defined as a type of parallel and distributed infrastructure which allows the geographically distributed autonomous and heterogeneous resources to be shared, selected and aggregated dynamically depending on their availability, capability, performance, cost, and user's quality-of-service requirements. This infrastructure offers to its users the same processing capabilities provided by supercomputers by creating a virtual supercomputer from connecting various networked and loosely coupled computers together allowing their resources to be shared among users. Computers, processing elements, software applications, printers, network interfaces, storage space and data are examples of resources. Middleware, computer software which provide basic services for resource management, security, monitoring, and so forth, are used to connect all these resources to a network. Due to the fact that resources are owned by various administrative organizations, local policies are defined to specify what is shared, who is allowed to access what and when, and under what conditions. The grid architecture is based on the creation of Virtual Organizations (VOs), a set of rules defined by individuals and institutions to control resource sharing [57] [92]. By sharing some or all of its resources, a physical organization can be part of one or more VOs [59]. Grid Computing has been increasingly used by commercial and non-commercial clients as a utility for solving scientific, complex mathematical, and academic problems, as well as for diverse applications [58].

Grid computing has witnessed several developments since the introduction of its early definitions in [56, 59]. The developments are aimed at a better understanding of the grid issues through the enhancement of the grid infrastructure and middleware. To provide services to its users, the grid computing system performs several activities as illustrated in Fig. 3.1. Grid resources are signed in within one or more Grid Information Services (GIS). The Grid Resource Broker (GRB) receives users' requirements to run their applications. GRB then queries GIS and carries out several operations which include assigning appropriate resources to users' applications and monitoring their execution until they finish.

3.2 Grid computing architecture

The architecture of a typical grid system can be seen as a stack of four layers, which are depicted in Fin. 3.2. More precisely, these layers are: fabric, core middleware, user-level middleware, and applications layers. Each layer provides a specific service, as follows [23] [132]:

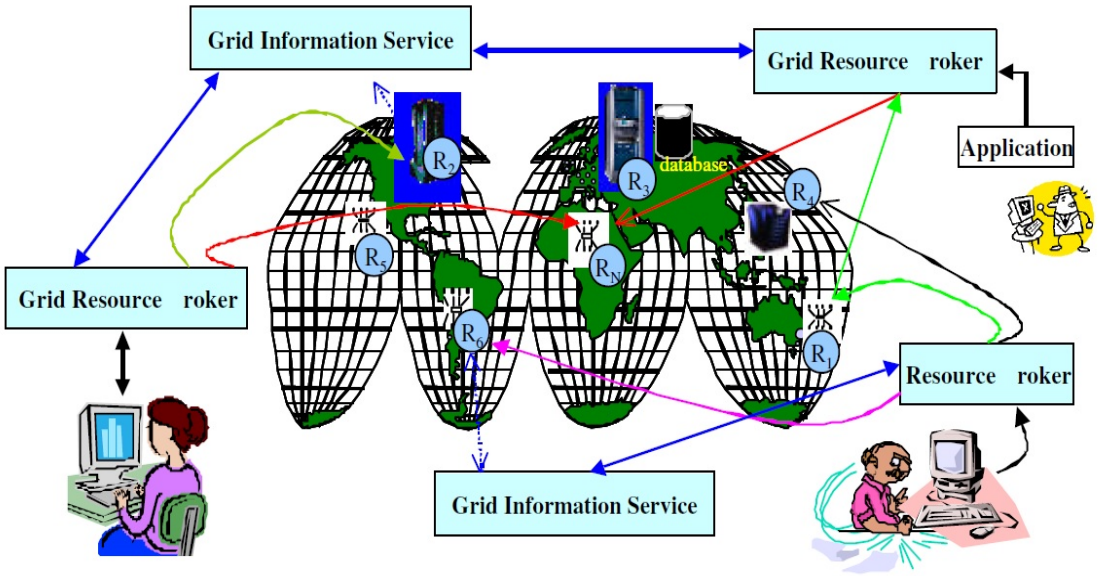

Fig. 3.1 A high-level view of a typical Grid computing system (adopted from [13]).

1. Fabric layer

   Also called the Physical layer, it contains distributed and autonomous resources such as computers, servers, networks, clusters, storage systems, data sources and other physical components. These resources are also heterogeneous as they have been coupled from different computing environments, each of which may use a different operating system (such as Windows and Linux).

2. Core middleware layer

   Also called the Connectivity layer, it aims at providing a unified environment, through the use of multiple interfaces, for the heterogeneous resources of the fabric layer. Resource monitoring, resource allocation, access to information and security are some examples of the services offered in this layer.

3. User-level middleware layer

   Also called the Resource layer, it offers services that provides a higher level of abstraction over the core middleware layer by utilizing its multiple interfaces. Examples include brokering (resource selection, management and aggregation), Debugging, programming languages and compilers.

4. Applications layer

   This layer provides interfaces which allow different users to submit their applications to the grid system and to collect results.

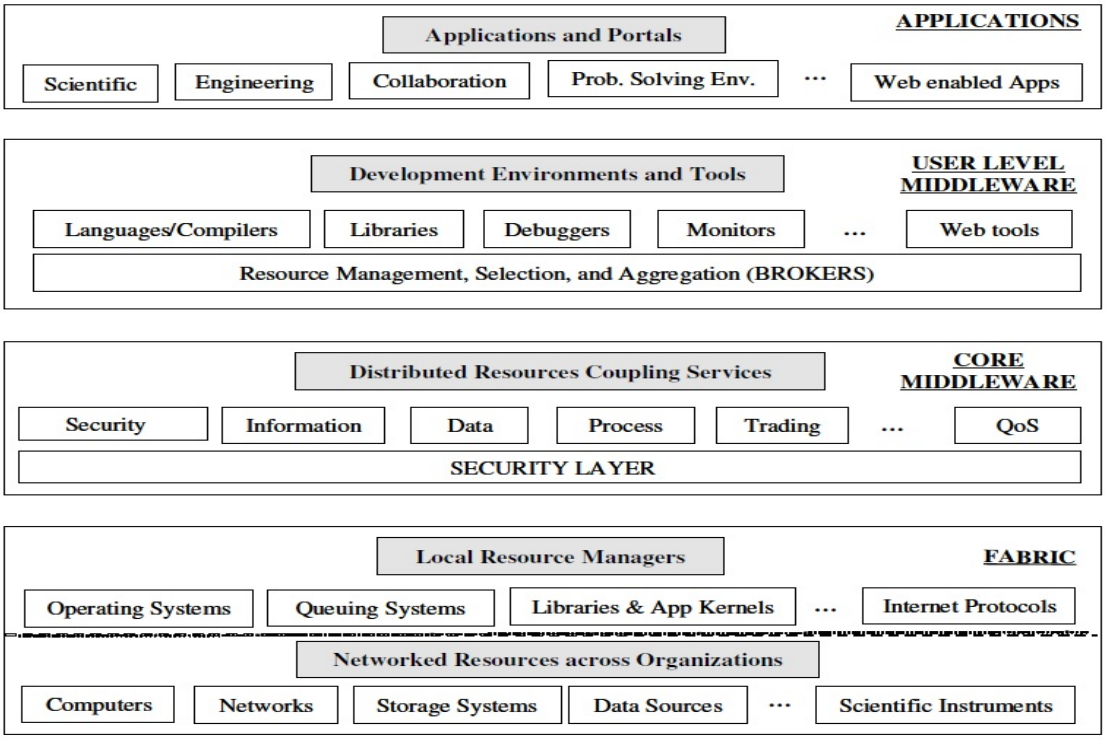

Fig. 3.2 The main layers and components of a typical Grid computing system (adopted from [23]).

3.3 Scheduling in grid computing

Allocating jobs, also called tasks or applications, to computational grid resources in an efficient manner is one of the main challenges facing any computational grid system; this allocation is called job scheduling in grid computing. An efficient scheduler is one which can make practical and effective use of the available distributed resources. These resources are connected through heterogeneous environments in an efficient, reliable and secure manner. Similar to job scheduling in traditional computing systems, this allocation is known to be an NP complete problem [161]; however, it is made more complicated in grid computing due to its dynamic nature, high degree of task and machine heterogeneity, problem size, and other factors such as existing local schedulers and policies [25]. To evaluate the job scheduling performance, an objective function should be defined such as maximizing resources utilization, minimising the makespan, and maximizing load balancing [124]. The scheduler's efficiency strongly depends on the algorithm applied to do the scheduling. Different algorithms could be used to do the scheduling which vary from simple heuristic methods to meta-heuristic methods. However, to enhance the overall performance of the grid, the meta-heuristic approaches are more likely preferred [117].

Scheduling problems in grid computing can be classified from different perspectives. According to the number of objectives to be optimized, the problem can be categorized into

single or multi-objective. The former includes one objective function such as maximizing resources utilization, minimising the makespan, or maximizing load balancing, whilst the latter consists of optimizing two or more objectives which are normally contradicted. The processing mode can also be used to classify scheduling problems into immediate or batch. The immediate scheduling, also called on-line scheduling, refers to the type of mapping in which jobs are scheduled once they arrive to the system. In the batch mode, jobs are collected in a group or a batch, then this group is scheduled. Therefore, better schedules are expected as this mode takes the advantages of knowing the characteristics of all submitted jobs and the available resources, which helps in making decisions about the job-resource mapping process [88].

Furthermore, scheduling might be categorised in terms of the interrelations between jobs into independent or dependent. In the first category, jobs are not related to each other, that is, inter-job dependencies are not available. In the second type, jobs cannot be divided as they should be processed using a predefined sequence, which means inter-job dependencies must be considered [55]. The independent job scheduling suits most of the characteristics of distributed heterogeneous environments such as grid computing systems. This is mainly due to the nature of their users, as different independent users submit their jobs and applications to be processed by these environments. Furthermore, the importance of the independent job scheduling arises in various realistic applications. Examples include those applications which use the SPMD (Single Program, Multiple Data) technique, such as data mining and image processing applications. Moreover, this type of scheduling is useful for applications that can be divided into independent parts such as Monte-Carlo simulations and parameter sweep applications [25] [85] [117].

Another possible classification is according to the type of environment, namely static or dynamic. In the static scheduling, the necessary information about jobs and resources are available in advance. These information do not change during the mapping process. In addition, no jobs are expected to arrive at the system after the allocation is performed [118]. Obviously, an accurate estimation of the processing time that each job requires to be executed by each resource should be available. This estimation can be provided by carrying out job profiling and statistical analysis of both submitted jobs and resource usage [162]. One major issue in the static scheduling is that resource failure and other unexpected events are not considered. However, additional mechanisms such as rescheduling can be added to deal in practice with these situations.

Static scheduling is useful in many different applications and domains. Predictive analyses, for instance, is one of the most common applications in which this type of scheduling is used to find an efficient mapping for some workloads sometime in the future, such as meeting a deadline, and to check in advance if the available time and resources are sufficient to finish such a mapping. Furthermore, static scheduling may also be used in the analyses of the requirements of distributed computing systems in which static

scheduling can be employed, for example, to justify the advantages of adding another piece of hardware to these systems in terms of productivity. Moreover, after performing a dynamic job scheduling, static scheduling can be utilized to check the quality of the achieved schedules by evaluating the performance of the dynamic scheduler. In this perspective, static solutions can be used to study the behaviour of a dynamic scheduler in terms of resources selection [19] [135].

On the other hand, in dynamic scheduling, jobs and resources can be added and removed to the system at runtime. This provides an efficient way to cope with any unpredicted events such as resource failure. A scheduler of this type uses actual information instead of estimations to assign jobs to resources [163]. However, in case of any change in the environment, such as the addition or deletion of a job or a resource, the previous global optimal will change and the scheduler should update the current information accordingly to act to this change, which makes the optimisation problem more complicated and challenging as a track of the new global optimal is required [106] [160].

In this thesis, the static batch independent job scheduling problem version in grid computing is considered as we will see in the next section. Different hybrid meta-heuristic algorithms are proposed in Chapter 4 to tackle this problem in terms of minimising a single objective, the makespan. Furthermore, these algorithms have been modified by adding the rescheduling strategy to tackle the dynamic version of the problem. The dynamic version of the problem together with the modified hybrid algorithms are described in Chapter 6.

3.4 Job scheduling in grid computing: Problem formulation

To study the job scheduling problem under different types of heterogeneous environments, a model that simulates the processing time of jobs on resources is required. Since the early 2000s, a common model called the Expected Time to Compute (ETC) has been used for this purpose [117]. The ETC model, which was introduced in [8], provides a framework for testing the performances of different scheduling algorithms under various circumstances.

This model assumes that an accurate estimation or prediction of the size of each job, the computing power of each resource, and an estimation of the load of the resources are known in advance. Furthermore, an accurate estimation of the expected execution time for each job on each resource is computable or is otherwise assumed to be known beforehand. These assumptions are realistic since it is easy to collect information about the computation power of resources and the jobs requirements from specifications provided by the user, by predications or from historic data [169]; see [84] [141] [101] for more details about the methods used for calculating this estimation based on analytical benchmarking and job profiling. This estimation is represented in a two-dimensional array called the ETC,

where row k of the ETC array consists of the estimated execution times for job k on each resource. Similarly, column p of the ETC array contains the estimated execution times of resource p for each job. Therefore, ETC[i][j] indicates the expected execution time that job i needs to finish on resource j. It also assumed the ETC [i, j] entry may include the time required to move job i and any data related to it from their known source to resource j [19].

A description of the static batch independent job scheduling problem in grid computing under the ETC model can be formulated as follows:

1. A set of n independent jobs J=$\{j_0, j_1, j_2, ..., j_{n-1}\}$ to be assigned to grid resources. Any job can be handled by any resource. However, these jobs are non-preemptive, i.e., each job should be executed entirely by one resource only.

2. A set of m heterogeneous machines R=$\{r_0, r_1, r_2, ..., r_{m-1}\}$ to be used for processing the submitted n independent jobs.

3. The ETC matrix is of size n x m, where ETC[j][r] denotes the estimated required time for processing job j by resource r.

4. The goal of job scheduling in grid computing is to find a mapping of submitted jobs to available resources that minimises the makespan, which itself represents the finishing time of the latest task and can be computed by Equation 3.1.

$$makespan = min_{s \in S} max_{j \in J}(Finish_j) \tag{3.1}$$

where S is the set of all possible solutions and $Finish_j$ represents the time by which job j is finished [85].

3.5 ETC matrix generating

The ETC matrix can be simply generated by dividing the size of a job by the computing power of a resource. One example of this type is the dataset of Liu *et al.* [93], publicly available from http://dx.doi.org/10.13140/RG.2.2.10787.04649, which consists of four instances of different sizes. The authors used the notation (the number of resources, the number of jobs) to describe each instance. The resource job pairs vary from small-scale instance (3, 13) to large-scale instances, such as (5, 100), (8, 60) and (10, 50).

However, to capture the various characteristics of grid computing environments, two methods, which are range-based and coefficient of variation (described in [8]), have been proposed to generate ETC matrices.

Each method defines three different types of metric, namely consistency, job heterogeneity, and resource heterogeneity. An ETC matrix is said to be consistent if a resource

R_a can process a job J_x faster than a resource R_b, then the same is true for any job J_k. If this structure is not maintained, i.e., R_a is not always faster R_b, then the ETC matrix is considered inconsistent. A mix of these two scenarios is the semi-consistent ETC matrix, which can be defined as an inconsistent ETC matrix with a consistent sub-matrix. Job heterogeneity indicates the degree to which the job processing times vary with two values, high or low. Similarly, resource heterogeneity models the degree to which the resource processing times vary for a given job and also has two values, high or low. Therefore, twelve distinct ETC matrices are needed so that we can consider all these various characteristics.

Both methods follow the same design structure to generate ETC instances. Although the coefficient of variation method allows for greater control over job and resource heterogeneity through the use of empirical probability distributions, it uses a more complex procedure than the range-based method [117]. Two ranges are used by the range-based procedure to generate ETC problem instances, namely [1, R_{job}] and [1, $R_{resource}$] for job and resource heterogeneity, respectively. For every job x, the method first generates a random number, Job, by sampling a uniform distribution from the first range. Similarly, the method generates another random number, Res, from the second range for every resource y. Then, the rows of ETC matrix are constructed by multiplying Job by Res [8]. The main steps to generate ETC problem instances using the range-based method are listed in algorithm 3.1.

Algorithm 3.1: The range-based procedure for generating ETC instances.

```
1 let n and m be the number of jobs and resources respectively;
2 for (x = 0 to n-1) do
3 |   Job ← a uniformly distributed random number in the range [1, R_job];
4 |   for (y=0 to m-1) do
5 |   |   Res ← a uniformly distributed random number in the range [1, R_resource];
6 |   |   ETC[x, y] ← Job * Res;
7 |   end
8 end
```

For realistic heterogeneous computing systems such as computational grids, the authors in [8] suggested typical values for R_{job} and $R_{resource}$ which are reported in the first row of Table 3.1. However, their work did not create a specific benchmark. The authors in [19] used the range-based method of [8] to generate the 12 classic ETC problem instances, which are publicly available from https://www.fing.edu.uy.

These instances, which are known as the Braun *et al.* dataset, have become a *de facto* standard benchmark to evaluate the performance of various scheduling approaches in heterogeneous environments and grid systems. However, they did not use the suggested values for R_{job} and $R_{resource}$ in [8]. Instead, the authors used the values listed in the second

Table 3.1 Job and resource heterogeneity parameters.

	job heterogeneity		resource heterogeneity	
ETC model	low	high	low	high
Ali *et al.*	R_{job}=10	R_{job}=100000	$R_{resource}$=10	$R_{resource}$=1000
Braun *et al.*	R_{job}=100	R_{job}=3000	$R_{resource}$=10	$R_{resource}$=1000

row of Table 3.1. Each instance has 512 jobs and 16 resources. The following abbreviation has been used to identify the type of ETC matrix, D-T-JHRH.0, where:

- D denotes the probability distribution type.
- T denotes the consistency type, with the following acronyms: c for consistent, i for inconsistent, and s for semi-consistent.
- JH denotes the heterogeneity of the jobs, with two possibilities either hi for high or lo for low.
- RH denotes the heterogeneity of the resources, with two possibilities either hi for high or lo for low.

The authors in [117] have reviewed the existing benchmarks in the literature. They came to the conclusion that none of the available datasets can actually simulate the current characteristics of grid computing systems in terms of the dataset size. Therefore, they proposed a new benchmark, known as the Nesmachnow *et al.* dataset, which is publicly available from https://www.fing.edu.uy, to cover larger cases. Each case consists of 24 instances which have been generated using the range-based method described in Ali *et al.* [8]. However, the first 12 instances are generated using the job and resource heterogeneity parameters described in Ali *et al.* [8], which are listed in the second row of Table 3.1; the other 12 instances are generated using the parameters proposed by Braun *et al.* [19], which appear in the first row of Table 3.1. The notation M-d-t-jhrh was used to describe each instance where M indicates the parameters used to generate the instance. The first 12 problem instances were generated using the proposed values of Ali *et al.* [8] and therefore the letter A is used to denote them, while the parameters employed by Braun *et al.* [19] were used to generate the second twelve instances and hence the symbol B is used to represent them. The d-t-jhrh notation follows the above description.

In this work, all the three datasets, namely Liu *et al.*, Braun *et al.* and Nesmachnow *et al.*, will be considered to test the performance of the proposed methods.

3.6 Methods for job scheduling in grid computing

The problem of job scheduling in grid computing is known to be an NP-Hard combinatorial problem, and hence, the exact methods are not applicable [65]. Alternatively, deterministic heuristic algorithms and meta-heuristic algorithms were used to address this problem. However, to effectively deal with its complexity, meta-heuristic algorithms are preferred [85].

Although they might not provide the best solutions, heuristic algorithms are preferred due to the fact that they can be easily implemented and provide solutions in a short period of time. Moreover, and for the same reasons stated above, the literature shows that most of them were used to generate solutions which are used to seed the initial population in various meta-heuristic algorithms, which reduces the time that these algorithms require to find satisfactory solutions as they start from a good point on the state space of the problem [118].

On the other hand, meta-heuristic algorithms are well-known approaches which have been applied effectively to a wide range of NP-hard problems. In fact, these algorithms are considered the best candidate in practice to cope with the complexity of job scheduling in a computational grid, and accordingly several algorithms have been suggested [124].

In the following subsections, a review of the heuristic and meta-heuristic algorithms described in the literature, will be presented.

3.6.1 Heuristic methods for job scheduling in grid computing

Several deterministic heuristic algorithms are described in the bibliography. In reference [19], Braun *et al.* described eleven heuristics when they tackled the job scheduling problem in heterogeneous environments. A more detailed study is the work of Kowk *et al.* in reference [91] which provided a comprehensive description of twenty seven heuristics for the static job scheduling problem. Recently, newer heuristics for the heterogeneous environments such as grid computing were proposed by Rafsanjani *et al.* and Gogos *et al.* in references [131] and [65], respectively.

This subsection explains some of the popular *ad hoc* heuristic methods for mapping independent static jobs to available resources in grid computing with respect to minimising the makespan.

- **Opportunistic Load Balancing (OLB) algorithm**

  A simple heuristic that maps each job in an arbitrary order to the next available resource without taking into account the required ETC to process each job on that resource. OLB keeps all resources as busy as possible, and hence, maximizes the resource utilization of the the whole system. However, it finds low-quality schedules

in terms of makspan mainly because it ignores the processing time each resource requires to execute each job [19] [100] [133].

- **Minimum Execution Time (MET) algorithm**

  The main strategy of MET is to assign each job in an arbitrary order to the resource with the minimum ETC for that job (the resource that executes that job faster than others), irrespective of that resource's availability [64] [102]. As a result of applying the above strategy, a high load imbalance across resources is expected which makes MET inapplicable in grid computing systems with consistent problem instances [19].

- **Minimum Completion Time (MCT) algorithm**

  MCT combines the advantages of OLB and MET [19]. The main idea of it is to assign each job in an arbitrary order to the resource with the minimum Completion Time (CT) for that job. The CT of job J on resource R is simply ETC[J, R] plus the current load of R, i.e., MCT uses ETC and resource loads to select the next assignment [135] [19].

- **Min-min algorithm**

  The min-min heuristic begins by calculating the minimum CT for all jobs and resources. It then determines the job j with the minimum CT and assigns it to the resource that obtains it. After allocating the job j, the CT matrix is updated. The same steps are repeated until all jobs are assigned [77] [19].The pseudo-code for the min-min heuristic is illustrated in Algorithm 3.2.

Algorithm 3.2: The min-min algorithm

```
1 For every job in the job set, calculate the completion time (CT);
2 jobs_removed ← 0;
3 while (jobs_removed<total number of jobs) do
4 |   Find the job i in the job set with the earliest completion time and the resource j
  |     which obtains it;
5 |   Assign i to j;
6 |   Delete i from the job set;
7 |   jobs_removed ← jobs_removed + 1;
8 |   Update the ready time and the completion time (CT) of resource j;
9 end
```

- **Max-min algorithm**

  The max-min algorithm follows the same steps of the min-min heuristic. It begins by calculating the minimum CT for all jobs and resources. It then determines the job j with the maximum CT and assigns it to the resource that obtains it. After

allocating job j, the CT matrix is updated. The same steps are repeated until all jobs are assigned [19].

- **Duplex algorithm**

  Duplex combines the Min-min and Max-min algorithms. It runs both of algorithms, then the best solution of them is selected [19]. Therefore, Duplex can be used to cope with the problems of the two previously described heuristics by utilizing the cases in which either Min-Min or Max-Min works better [73].

- **Longest Job to Fastest Resource – Shortest Job to Fastest Resource (LJFR-SJFR) algorithm**

  Similar to the min-min heuristic, LJFR-SJFR begins by calculating the minimum CT for all jobs and resources. Let t_j be the total number of jobs and let t_r be the total number of resources. LJFR-SJFR consists of two steps. The first step involves the assigning of the t_r longest jobs to the t_r available resources. The second step includes the mapping of the shortest job to the fastest resource, and the longest job to the fastest resource alternatively. The current load of each resource is updated after each assignment. The second step is repeated until all t_j - t_r are allocated [1] [79].

- **Sufferage algorithm**

  The main idea of this heuristic is to use the fact that a job *suffers* if it is not assigned to the resource that can process it faster than all the available resources [65]. This heuristic uses the difference between the first and second best jobs in terms of MCT to compute the suffrage value. It then allocates jobs with high suffrage values to the resources that can process them at the earliest time [11].

3.6.2 Meta-heuristic methods for job scheduling in grid computing

The problem of job scheduling in distributed and heterogeneous computing environments, such as grid computing, has been addressed using different approaches such as simple queuing and heuristic algorithms. However, to effectively deal with the associated complexity, meta-heuristic algorithms are preferred [85]. Meta-heuristic algorithms are well-known approaches which have been used effectively to solve a wide range of NP complete problems. In fact, these algorithms are considered the best candidate, in practice, to cope with the complexity of job scheduling in grid computing, therefore, several algorithms have been suggested [124].

3.6.2.1 ACO for job scheduling in grid computing

ACO is one of the meta-heuristic search methods which simulates the behaviour of ants in foraging for food [38], that has been used to address the job scheduing problem in grid computing.

A loosely coupled hybrid ACO algorithm was suggested by Ritchie *et al.* [135] which combines ACO with TS to improve the performance of a number of similar approaches proposed in [19]. To encode information in the pheromone trail, the MCT heuristic was used. Furthermore, pheromone trails were updated using the Max-Min Ant System (MMAS) rule proposed in [146]. The authors used the classical Braun *et al.* [19] dataset to evaluate their proposed scheduler. Their experimental results demonstrated that the hybridization of TS with ACO improved the makespan of the solutions. However, the hybrid method needed over 3.5 hours to achieve these results.

Fidanova and Durchova [54] proposed a Monte Carlo ACO-based algorithm for solving the static independent batch job scheduling problem in grid computing systems in terms of minimising the makespan. A new heuristic function called *free(r)* was introduced, which indicates the time by which the resource r will be free. This heuristic provides information about which resources are released earlier. Obviously, a resource will be more desirable if it is free earlier. In this algorithm, the objective function used by authors was the maximum value of the free function which has been recorded during solution construction. Three simple grid scenarios were used to evaluate the proposed ACO-based scheduler, each of which consists of 20 jobs and 5 resources. One ant only was used by the authors and the ACO performance was compared with the on-line heuristic in which the former method outperformed the later in all three cases.

An ACO-based scheduler for dynamic job scheduling in grid computing was proposed by Lorpunmanee *et al.* [94]. Minimization of the total job waiting time was the main goal of the proposed scheduler which consists of four steps. The proposed scheduler used local update and global update rules to update the pheromone value on each resource. In addition, the scheduler used the Completion Time (CT), which is the time a machine needs to finish executing a job measured as clock time. The authors defined a grid environment in which jobs arrive to the system at different times, the availability of resources is regularly changing, one job could be processed by each processor per unit time and jobs are independent of each other. In the study, the performance of ACO based scheduler was compared with First Come First Serve (FCFS), Earliest Due Date (EDD) and Earliest Release Date techniques (ERD). A discrete-event grid simulator, called GridSim toolkit [22], was used to develop the proposed method in which problem instances of up to 3000 jobs and 20 resources were generated to evaluate the performance of their proposed meta-heuristic. The experimental results showed that ACO achieved the best results compared to the other approaches explored in the study.

Kousalya and Balasubramanie [89] studied the hybridization of ACO meta-heuristic with five new local search methods in a loosely coupled fashion for solving the static independent batch job scheduling problem in grid computing in terms of minimising the makespan. That is, the output of ACO is further improved by one of the five local search methods, resulting in five new loosely coupled hybrid meta-heuristics. The move and swap concepts were used to design these five local search methods. The authors used the standard benchmark problem instances of Braun et al. [19] to carry a number of experiments to evaluate the performance of the suggested methods. The experimental results illustrated that hybridizing ACO with local search methods provides better schedules than the stand-alone ACO. Moreover, the reported results encouraged the authors to suggest studying the behaviour of these hybrid methods in the dynamic mode.

A more effective ACO-based grid scheduling algorithm was introduced by Mathiyalagan *et al.* [105]. The developed scheduler has modified the original ACO algorithm presented in [38] by changing the basic pheromone updating rule. The GridSim toolkit [22], was used to develop the proposed method in which problem instances of up to 100 independent jobs were generated to evaluate the performance of the proposed modified ACO. The experimental results showed that this modification increased efficiently the algorithm performance in terms of makespan compared to the original ACO.

MadadyarAdeh and Bagherzadeh [96] approached the static independent batch job scheduling problem in grid computing as a single-objective optimisation problem that minimises the makespan. To tackle this problem, an ACO-based scheduler that uses the heuristic function of Fidanova and Durchova [54] was developed. However, the authors proposed a modified probability rule, in which the standard deviation of the jobs is included. This rule is required to select the next job-resource mapping. To evaluate the performance of the proposed algorithm, four small cases, each of which has 32 jobs and 4 resources, were considered. The reported results showed that this modification allowed the proposed algorithm to improve its performance in terms of makespan compared to the original ACO.

Ku-Mahamud *et al.* [90] proposed a modified ACO-based meta-heuristic for solving the static independent batch job scheduling problem in grid computing systems in terms of minimising the makespan and utilization. The new ACO-based scheduler used a modified procedure to update pheromone levels that is applied in two cases. The first case includes local update when any improvements occur (on the iteration level), while the second case involves applying pheromone updating globally using the best solution found so far. Simple grid examples of up to 100 jobs and 7 resources were used to evaluate the proposed modified ACO-based scheduler. A population of 7 ants only was used by the authors and the ACO performance was compared with the non-modified ACO in which the former method outperformed the later in most of the cases.

Christina and Miriam [28] suggested a multi-objective ACO-based meta-heuristic algorithm, called MCACO, for solving the static independent batch job scheduling problem

in grid computing in terms of minimising two objectives, namely the makespan and flowtime. For each job and resource, the authors used an indicator, called PV, to represent them, which combines several features, such as the job size, resource speed, current resource load, the expected completion time of a job on a resource and the available bandwidth between a job and a resource. Therefore, a two-dimensional array, $PV[j,r]$, is first constructed, then the highest $PV[x,y]$ value is identified. Scheduling is then performed by allocating the job x to the resource y. The allocation step is followed by a pheromone update to save information about the current status of the resources for the next iterations. The GridSim toolkit [22], was used to develop the proposed method in which small problem instances of up to 30 independent jobs and 40 resources were generated to evaluate the performance of the proposed modified ACO. The experimental results showed that the MCACO algorithm outperformed the original ACO in terms of makespan and flowtime in all of the cases.

A loosely coupled hybrid algorithm, which combines ACO and GA, i.e. ACO+GA, was suggested by Alobaedy and Ku-Mahamud [10] as a promising algorithm to minimise the makespan and flowtime in computational grids. ACO starts first and the output of it will be used by GA which further improves it. The ETC model was used to evaluate the proposed method by generating a special 512x16 dataset using the range-based method proposed in [8]. The proposed hybrid method outperformed the other stand-alone meta-heuristics explored in the work. However, their dataset and implementation are not available to make a fair comparison.

In a recent survey, Oshin and Chhabra [123] reviewed the use of ACO for job scheduling in grid computing. Different parameters were used in the survey to present an analytical study of variants of ACO-based schedulers for static and dynamic independent batch job scheduling in grid computing systems. Based on the reviewed papers described in the literature, the authors drawn several conclusions. One conclusion includes the use of ACO for job scheduling in grid computing in which they stated that ACO meta-heuristic is one of the best candidates to address this problem. However, the performance of the existing ACO-based schedulers faced one problem, as the authors observed, which is the time ACO-based algorithms take to start constructing good schedules as the pheromone trail is accumulated after performing a few iterations to identify the best job-resource mappings. One way to resolve this problem, as the authors suggested, is to hybridize ACO with other meta-heuristics.

3.6.2.2 GA for job scheduling in grid computing

ACO is not the only approach; the literature includes many other meta-heuristic algorithms such as the GA. A GA is a meta-heuristic search method that mimics the evolution of living

beings. It has been successfully used for solving many NP-hard optimisation problems closely related to the job scheduling in heterogeneous environments and grid computing.

The primary work presented by Tirat-Gefen and Parker [154] suggested a tool, called MEGA, to design heterogeneous multiprocessor systems and minimise the processing time of a given set of dependent jobs that are represented using a Directed Acyclic Graph (DAG). A component, called MILP (Mixed Integer Linear Programming), in the mathematical model of a tool set, called SOS (Synthesis of Multiprocessors Systems), was used to develop MEGA. In the proposed tool, a GA was used together with a fast linear-time algorithm to check that the achieved job schedule satisfies the timing constraints. However, in heterogeneous environments such as grid computing systems, makespan is the most common performance-related metric used since it represents the productivity (throughput) in such systems [65].

Shroff *et al.* [140] proposed a strongly coupled hybrid meta-heuristic, called GSA (Genetic Simulated Annealing) algorithm, which combines two meta-heuristic methods, namely GA and SA, to solve the problem of dependent job scheduling in heterogeneous environments in terms of minimising makespan. In the proposed method, the usual genetic operators (selection, crossover and mutation) are applied. However, GSA used the temperature cooling concept of SA to perform reproduction. The main reason behind this, as the authors claimed, is to maintain diversity in the population. The performance of GSA was evaluated using randomly generated datasets which consists of 10-100 jobs and 5-35 resources. Although the datasets that were used are small, the reported results show that GSA is able to find good quality solutions in a reasonable time.

Wang *et al.* [156] considered the problem of dependent job scheduling in heterogeneous environments to which a new GA approach was proposed that minimises the makespan. Two important aspects can be marked in their work. The first aspect includes the use of estimated completion time to verify the performance of the proposed approach. Secondly, an *ad hoc* heuristic, called baseline, was used to seed the initial population of the proposed GA. These two aspects were adopted by many subsequent works resulting in various successful meta-heuristic schedulers which were used to tackle diverse scheduling problems. Along with this seeded solution, a non-clustering rule (described in Section 2.5.2), which prevents the generation of identical individuals, and hence, avoid trapping the GA in a premature convergence, was used to generate the remaining solutions of the initial population. Furthermore, several genetic operators were used to validate the proposed GA-based scheduler. To evaluate the performance of the proposed approach, the proposed method was compared against several heuristic algorithms using datasets that contain up to 100 jobs and 20 resources. The experimental results showed that the proposed GA achieved better solutions than the other heuristic algorithms used in the work.

Abraham *et al.* [1] suggested three nature-inspired meta-heuristic algorithms, which are GA, SA and TS, for static independent job scheduling in grid computing in terms

of minimising two objectives, namely the makespan and flowtime. Furthermore, the authors used the above algorithms to suggest two hybrid algorithms. The first hybrid algorithm, called GA-SA, combines GA and SA in a strongly coupled fashion in which the LJFR–SJFR heuristic was used along with the random method to generate the initial population, then instead of applying the normal genetic operators, GA-SA applies the cooling and reheating concepts of SA to assign jobs to resources. This step is followed by a feasibility check procedure in terms of resource availability and user specified requirements. The second hybrid algorithm, called GA-TS, combines GA and TS in a strongly coupled fashion in which TS was used as a mutation operator. Although this work was one of the pioneering in terms of proposing hybrid meta-heuristic algorithms to job scheduling in grid computing, the experimental results that were reported include the application of GA only to a small dataset that consists of 13 jobs and 3 resources, i.e., the authors did not test the performance of SA, TS, GA-SA and GA-TS.

Braun *et al.* [19] extended the works of Wang *et al.* [156] and Shroff *et al.* [140] when they adopted their GA-based approaches, together with other nine heuristics described in the literature, to solve the static independent job scheduling problem in heterogeneous environments (in terms of minimising the makespan). The min-min heuristic [77] was used to seed one individual of the initial generation that consists of 200 solutions, while the remaining 199 individuals were generated randomly. To evaluate the performance of the adopted approaches, the ETC model was used in which 100 instances were generated each of which has 12 cases. The first instance, which is known as the Braun *et al.* dataset, has become a *de facto* standard benchmark to evaluate the performance of various scheduling approaches in heterogeneous environments and grid systems. Their experiments show that the adopted GA achieves better results than the other 10 algorithms explored in the work.

On the other hand, Theys *et al.* [152] extended the works of Wang *et al.* [156] and Braun *et al.* [19] when they applied the eleven approaches described in [19] to tackle three different types of scheduling problems in heterogeneous environments. The first type included the static scheduling of dependent jobs. The second version considered the dynamic scheduling of dependent jobs in which the rescheduling technique was used to introduce dynamism. The third case examined the static scheduling of independent jobs. For all scheduling cases, their experiments show that the adopted GA achieves better results than the other 10 algorithms explored in the work, suggesting that the GA can be used successfully to tackle different types of scheduling problems.

A strongly coupled meta-heuristic which combines a GA with a heuristic method, called list scheduling, was proposed by Grajcar [67] to solve a version of scheduling in which a partially ordered set of jobs are to be mapped to a heterogeneous multiprocessor system aiming at minimising the makespan as well as satisfying some resource usage constraints and data dependencies. After applying the usual genetic operators, the proposed method calls the list scheduling heuristic to evaluate the new offspring then it applies a

steady-state reproduction to generate the new population. To verify the performance of the hybrid GA, the author compared the performance of the proposed hybrid GA with some exact methods described in the literature using several instances with up to 96 jobs. The experiments show that the strongly coupled hybrid GA obtained the optimal solution for problem instances in significant reduced execution times compared to the selected exact methods. Subsequently, Grajcar [68] investigated the use of the strongly coupled GA scheduler for dependent job scheduling in heterogeneous computing systems. The author highlighted a major weakness in the proposed hybrid algorithm that comes from the lack of information about jobs which are not scheduled yet. Three approaches were presented to deal with this weakness. However, the proposed strongly coupled hybrid meta-heuristic failed to find the optimal solutions in two cases, suggesting a further generalization of the hybrid algorithm is required to be explored.

Zomaya and Teh [179] considered the dynamic variant of job scheduling problem in parallel and distributed computing systems in which several objectives are optimized at the same time. These objectives include minimising the makespan, maximizing the resource utilization and maximizing the load-balancing. To tackle this problem, a centralized GA-based scheduler was proposed. The proposed algorithm used a two-dimensional direct representation to encode the problem. A technique, called sliding-window, was used to generate the initial population. The centralized GA adopted the following genetic operators: roulette wheel selection, cycle crossover and swap mutation. The suggested GA was allowed to run for k generations, k=10. Different problem instances of up to 1000 jobs and 50 resources were used to run several experiments. Two methods were used to study the performance of the proposed method, namely the First Fit heuristic and a random assignment scheme. The experimental analysis showed that GA provided better schedules than the other methods in terms of makespan and resource utilization. As the number of jobs increases, an almost full resource utilization was achieved by the the proposed GA-based scheduler. On the other hand, as the number of resources increases, better makespan values and low average resource utilization were reported.

Page and Naughton [126] extended the work of Zomaya and Teh [179] by developing a dynamic GA-based scheduler for heterogeneous computing environments that considers variable system resources. The dynamic GA used an initial population of 20 individuals which was generated using the list scheduling heuristic. The following genetic operators were used: roulette wheel selection, cycle crossover and two types of mutation operators, namely random swap and re-balancing. The suggested GA was allowed to run for 1000 generations. A randomly generated dataset that consists of 10000 jobs and 50 resources was used to run several experiments. The performance of the proposed algorithm was compared with five selected heuristics described in the literature and with the GA of Zomaya and Teh [179]. The reported results showed that the proposed GA consistently outperformed the other methods explored in the work.

A hybrid algorithm for the dynamic scheduling of some directed graph-based workflows in grid computing systems was proposed by Prodan and Fahringer [129] as a part of the ASKALON project. The proposed hybrid algorithm was based on combining a GA with some classical static DAGs scheduling heuristic techniques that were iteratively invoked during the runtime. These classical techniques were generated using well-defined job migration and cycle elimination methods. An experiment management tool especially designed to study the performance of parallel applications, called ZENTURIO [128], was used to implement the proposed hybrid method. The authors successfully implemented a real-world application to which they reported their experimental results which showed that the hybrid GA was quite effective in obtaining good static and dynamic schedules within large complex heterogeneous environments.

Sugavanam *et al.* [149] considered a constrained static job scheduling problem in heterogeneous environments in which the robustness of a schedule is to be maximized, i.e, jobs are statically allocated to resources and the makespan of their schedule should not exceed a predefined limit. Seven methods were proposed, which include heuristic and meta-heuristic algorithms. The heuristic methods were Max-Max, GIM (Greedy Iterative Maximization), OIM (Overhead Iterative Maximization), while the meta-heuristics were GENITOR, MA (Memetic Algorithm), ACO, and HEA (Hereboy Evolutionary Algorithm). GENITOR [158] is a variation of the GA. The Max-Max heuristic was used to seed one individual to the initial population that consists of 200 individuals; the remaining 199 were generated randomly. To provide a specific selective pressure, a special function, called the linear bias function, was used to select parents. The other genetic operators include: one point crossover and random move mutation. A maximum number of generation of 250000 was used as a stopping condition. On the other hand, the proposed MA combined GENITOR with a local search method (hill climbing) in a strongly coupled fashion. MA follows the same steps of GENITOR with the exception that the hill climbing method is applied at three places of the algorithm. These places are: after generating the initial population, after performing crossover and after performing mutation. Nonetheless, HEA is a single-individual fast evolutionary algorithm which combines the features of GA and SA. Since it works with only one individual, HEA applies mutation only, i.e, it does not perform any crossover and selection operations. The proposed ACO was adopted from Ritchie *et al.* [135] described in the previous subsection. A dataset of 1024 jobs and 8 resources was used to evaluate the seven methods. The reported results showed that the proposed OIM heuristic consistently outperformed the other methods explored in the work. GIM, GENITOR and MA achieved results that are within 2 percent of OIM, however, more times are required to obtain their results. Among all, HEA performed the worst.

Subsequently, Sugavanam *et al.* [148] extended their previous work by adding a second variation of the scheduling problem that involves the selection (purchasing) of a fixed set of resources, within a given budget constraint, to construct a heterogeneous computing

system. The authors suggested a new heuristic, called CPI-SIM (Cost Performance Index Sum Iterative Maximization), which was used along with the other methods described in their previous work (with the exception of OIM and ACO) to tackle the problem. A dataset of 1024 jobs and 33 resources was used to evaluate the six methods. The reported results showed that the proposed GENITOR heuristic consistently outperformed the other methods explored in the work. CPI-SIM and GIM achieved results that are within 2 percent of GENITOR, using almost the same runtime. Among all, HEA and MA performed the worst.

Carretero and Xhafa [24] explored the use of two GA variants for developing an efficient multi-objective scheduler which assigns jobs in static and dynamic large scale grid applications in terms of minimising the makespan and flowtime. The first variant was a hierarchical GA in which the two objectives are optimized by their importance, while the second type was a simultaneous GA in which the makespan and flowtime objectives are optimized simultaneously. Two types of encoding schemes were used, namely the direct and permutation-based encodings. Two *ad hoc* heuristics, which are the MCT [100] and the LJFR−SJFR [1], were employed along with the random approach to seed the initial generation in order to introduce diversity. The main reason behind selecting the LJFR−SJFR heuristic was its capability to simultaneously minimise both makespan and flowtime (the makespan is minimised through the LJFR, while the flowtime is minimised by the SJFR). Several genetic operators for both encodings were carefully examined and tuned aiming at identifying which of them best suits the problem. To simulate realistic grid computing systems, a simple grid simulator was developed by the authors which is able to generate large scale problem instances to be used in the evaluation of the proposed GA-based schedulers. As a stooping condition, the authors used a fixed time of 90 s. The reported results are twofold. The first part provided the results of applying the proposed GAs to the static dataset of Braun *et al.* [19], to which the simultaneous GA outperformed the results achieved by Braun *et al.* [19] in 11 out of 12 instances. The second part included the results of applying the proposed methods to the dynamic version of the problem in which the rescheduling technique was used to introduce dynamism. The experimental results showed that the Steady-State simultaneous GA obtained the best results in terms of makespan and flowtime, suggesting the GA as powerful meta-heuristic scheduler for grid computing systems. Later, Carretero and Xhafa [25] improved their work by using a weighted sum function that combines the makespan and flowtime objectives. After performing a preliminary tuning process, more priority was given to the makespan over the flowtime. The reported results include applying the proposed GAs to the static dataset of Braun *et al.* [19] only, to which the Steady-State simultaneous GA outperformed the results achieved by Braun *et al.* [19] and the author's previous work in 11 out of 12 instances. These two works were extended later by Xhafa and Carretero [159] in which a discrete event-based simulator, called HyperSim-G [27], designed to study the behaviour

of different algorithms in grid computing systems, was used to test the performance of the above methods in static and dynamic versions of the problem.

The use of a GA variant, called Struggle GA, for solving static independent job scheduling in computational grids in terms of minimising the makespan was investigated by Xhafa and Duran [166]. The Struggle GA, which was proposed by Grüninger and Wallace [70], is a Steady-State GA that has a special adaptive replacement operator. Unlike the Steady-State GA which replaces the worst solution, an old solution in the Struggle GA is replaced by the most similar solution to it that has a better fitness score. The goal of applying such a replacement strategy is to maintain certain diversity in the population. However, it requires high computational power and time as it includes additional comparisons to determine similarities between the individuals. Therefore, the main contribution of Xhafa and Duran [166] is to introduce an efficient replacement operator that is linear rather than exponential in terms of time complexity. To achieve this goal, the authors suggested three hash-based replacement operators and they referred to them as a key, b key and c key. The other genetic operators were exactly the same of those used in [24] and [25]. The reported results include applying the proposed Struggle GAs to the static dataset of Braun *et al.* to which the c key hash-based replacement Struggle GA outperformed the results achieved by the other hash-based keys replacement operators in 10 out of 12 instances.

Xhafa *et al.* [165] exploited the use of two hybrid cellular MAs (cMAs), cMA is a type of MA that uses a structured population, for solving the multi-objective static independent job scheduling problem in grid systems in terms of minimising the makespan and flowtime. The first hybrid method, called cMA + LMCTS, combines cMA and a local search method, called LMCTS (Local Minimum Completion Time Swap), in a strongly coupled fashion, while the second hybrid method, called cMA + LTH, integrates cMA and a local search method, called LTH (Local Tabu Hop), in a strongly coupled fashion as well. The call of the local search occurs after applying crossover and mutation operators. If there is an improvement, the old solution is replaced by the new one. It is worth to mention that the authors used a version of cMA that performs crossover and mutation separately. The LJFR−SJFR heuristic was used along with the random method to generate the initial population. Six different selection operators were examined and the tuning process showed that N-tournament provided the best results. Furthermore, several crossover operators were also studied, in which the one-point crossover performed better than the other operators. Moreover, four different mutation operators were tested, namely move, swap, move-swap, and re-balance, among which the re-balance mutation obtained the best results. The classical static dataset of Braun *et al.* [19] was used to verify the performance of the two proposed hybrid cMA-based schedulers. The results of the proposed methods were compared with three GA-based schedulers, namely the GAs of Braun *et al.* [19], Carretero and Xhafa [24] and Xhafa and Duran [166], respectively. The reported results showed that

the strongly coupled hybrid cMA + LTH meta-heuristic outperformed the other methods explored in the study in terms of minimising the makespan, while the other strongly coupled hybrid cMA + LMCTS meta-heuristic obtained the best results in terms of flowtime. The achieved results motivated the authors to suggest exploring the application of the proposed methods for the dynamic version of the problem. Moreover, these results highlighted the robustness of strongly coupled hybrid meta-heuristic methods for job scheduling problems in grid computing systems. Later, Xhafa [160] and Xhafa *et al.* [168] extended this work by suggesting more local search methods (16 methods) and exploring the resulted strongly hybrid MA meta-heuristic algorithms for the dynamic version of the problem to which the author used the HyperSim-G [27] grid simulator to test the performance of the proposed methods. Among the 16 methods and for the static and dynamic versions of the problem, MA + TS provided the best results in terms of minimising the makespan.

Two hybrid meta-heuristics were proposed by Xhafa *et al.* in [167] and [169] to address the task scheduling problem in computational grids. The former method combines a GA and TS in a strongly coupled fashion in which the GA is the main algorithm which calls TS during its execution to further enhance the quality of the solutions in the population. On the other hand, the latter method combines the same methods in a loosely coupled fashion, that is, the GA is executed first and its final solution is further improved by TS. The performance of both proposals was evaluated using the HyperSim-G grid simulator [27]. However, the first work addressed the static version of the problem, while the second tackled the static and dynamic versions. Later, Xhafa *et al.* [170] compared between the performance of them. The experimental results showed that the strongly coupled algorithm outperformed the other methods explored in the study for the small and medium sizes while the loosely coupled obtained the best makespan results for the large size problem instances. In general, the hybrid meta-heuristics achieved better makespan results than the stand-alone methods investigated in the study.

Falzon and Li [52] considered the problem of dependent job scheduling in grid computing environments to which an enhanced GA was proposed that minimises the makespan. Two important aspects can be marked in their work. The first aspect includes the employment of a heuristic method as a mutation operator, in which the authors used a heuristic method called TMWD (Task Matching with Data) described in [53]. Secondly, the authors evaluated the effects of seeding the initial population of the proposed GA with solutions from heuristics, in which they used the enhanced heuristic methods proposed in [53]. These two aspects were adopted by many subsequent works resulting in various successful meta-heuristic schedulers which were used to tackle diverse scheduling problems. The DAG data structure was used to model job workflows. To represent a solution, the authors used the encoding method proposed by Wang et al. [156]. Several selection techniques were examined, including the Elitism, Rank-based and Stochastic Universal Sampling. Additionally, various crossover operators were tested. To evaluate the performance of

the proposed methods, a dataset that consists of 100 schedules with up to 500 jobs, was generated using a simulation tool, called DAG Simulator, described in [51] [53].

A hybrid multi-objective meta-heuristic was proposed by Kardani-Moghaddam *et al.* [83] that combines a GA and VNS in a strongly coupled fashion to solve the static independent job scheduling in market-based grids in terms of minimising the cost and makespan. The GA worked as a main algorithm which during its execution calls VNS to further improve the quality of the individuals. After performing the usual genetic operators, the VNS meta-heuristic is applied on some individuals which are selected according to a specific probability. The reason behind not applying the VNS procedure to all individuals, as the authors explained, was due to the high computational cost, recalling that a population of 60 individuals was used. Two variants of min-min and Sufferage *ad hoc* heuristics, called MinCTT and SuffCTT described in [63], were used to generate the initial population along with the random method. The direct representation was used to encode the problem and the wighted sum function was used to combine the two objectives. For selection, crossover and mutation, the authors used Roulette Wheel, two-points and random move operators, respectively. For VNS, only one nighbourhood structure was used that is based on the concept of random moving in which the job with the highest execution cost is first identified then the resource to which it is assigned is replaced with another randomly selected resource. Moreover, the proposed VNS used a local search procedure that sorts the resources according to their local makespans, then the resources of some selected jobs with higher execution costs are replaced with resources that have low local makespans. To evaluate the performance of the proposed hybrid method, a random dataset was generated that has 500 jobs and 10 resources. The results obtained by GA−VNS were compared with the output of some heuristics described in [63]. The reported results showed that the strongly coupled hybrid GA−VNS achieved the best results compared to the other methods explored in the study.

Kolodziej *et al.* [87] approached the independent batch job scheduling problem in grid computing as a multi-objective optimisation problem that minimises the makespan and energy consumption. To reduce the power energy consumed by the resources, a mechanism called DVS (Dynamic Voltage Scaling) was used. To tackle this problem, two GA-based schedulers that use elitist and struggle replacement techniques were developed. To evaluate the performance of the proposed algorithms, four cases with different sizes in static and dynamic modes were considered. The reported results showed that a fair reduction in the energy usage and makespan was achieved by the suggested method.

Kolodziej *et al.* [86] developed a scheduling model which integrates the classical scheduling performance metrics, namely the makespan and flowtime, with job security requirements. Eight GA-based schedulers were proposed to tackle this multi-objective scheduling problem in static and dynamic modes. Two grid computing cases were considered, namely secure and risky, in which the HyperSim-G [27] grid simulator were

used to implement them and to generate small- to large-scale problem instances. Two heuristics, namely MCT and LJFR–SJFR, were used along with the random method to generate the initial population. For selection, crossover, mutation and replacement, the proposed methods used Linear Ranking, CX and PMX, Rebalancing and Steady-State, respectively. The proposed meta-heuristics are experimentally evaluated in static and dynamic grid scenarios by using a Grid simulator. A fast reduction in makespan and flowtime values, especially in the dynamic mode, were reported that confirms the usefulness of the suggested meta-heuristics in dynamic job scheduling. Later, Kolodziej *et al.* [88] extended their previous works by combing the classical scheduling performance metrics, namely the makespan and flowtime, with energy and risk requirements. Furthermore, the authors developed multi-population GA-based schedulers. Moreover, a two-model grid toolkit, called Sim-G-Batch, was developed that simulates the infrastructure of a grid computing in which different job scheduling scenarios can be performed.

A hybrid meta-heuristics, called HGAPSO, was proposed by Zahedani and Dastghaibyfard [177] to address the static independent batch job scheduling problem in computational grids in terms of minimising the makespan and flowtime. The proposed method combines the GA of Carretero *et al.* [25] and the Discrete PSO (DPSO) of Izakian *et al.* [80] in a strongly coupled fashion, in which the GA first performs the normal selection, crossover, and mutation operators, then DPSO is applied to further improve the quality of the solutions in the population. The classical static dataset of Braun *et al.* [19] was used to test the performance of HGAPSO and the experimental results showed that it outperformed the results achieved by the stand-alone methods investigated in the study, namely min-min, min-max and DPSO approaches, for most of the instances.

Oshin and Bhatt [122] proposed a loosely coupled hybrid algorithm that combines PSO, Cuckoo Search (CS) and GA for solving the static independent batch job scheduling in grid computing in terms of minimising the makespan and flowtime. A random initial population is first generated, then PSO starts mapping the received jobs to the available resources and constructs schedules. CS improves these schedules using the Levy flight concept and pass the improved schedules to the GA which further enhances it. To study the performance of the proposed hybrid method, the authors generated a simple dataset with up to 500 jobs and 50 resources. The results achieved by the proposed approach were compared with the results obtained by PSO, FCFS, SJFS and LJFS/FF methods, in which the proposed method outperformed them in all of the cases tested.

3.6.2.3 Other meta-heuristics for job scheduling in grid computing

Besides ACO and GA, the literature includes other meta-heuristic methods such as VNS, TS, CHC, PSO, DE and SA. VNS is based on the systematic change of the neighbourhood during the search process. The traditional VNS starts with an initial solution, then explores

its neighbourhoods and moves to a new one if an improvement is found. The exploration step is followed by a local search in order to move from solutions in the neighbourhood to a local optimum.

Davidovic *et al.* [31] developed two meta-heuristic algorithms, namely VNS and TS, for solving the static dependent multiprocessor scheduling problem in terms of minimising the makespan. The problem was represented using a DAG data structure. The proposed approaches used two *ad hoc* heuristics, which are CP and LS, along with the random method to generate the initial solution. Additionally, a simple swap neighbourhood structure was used, that randomly moves a job from one resource to another. A local search that applies all feasible neighbourhood structures and moves to a new neighbour if an improvement occurs was selected. A random dataset that contains up to 300 jobs and 8 processors was generated to run several experiments. The performance of the developed algorithms was compared with the CP and LS heuristics, the Multi-start Local Search (MLS) and a modified version of Problem Space Genetic Algorithm (PSGA) described in [5]. The experimental results showed that VNS outperforms them in all of the cases investigated.

A Multi-objective Variable Neighbourhood Search (MVNS) algorithm was proposed by Selvi and Manimegalai [137] to solve the static independent job scheduling problem in grid computing in terms of minimising both makespan and flowtime. The authors introduced five different neighbourhood structures and the random Problem Aware Local Search Heuristic (PALS) was used. The performance of the proposed method was compared with some of the methods discussed in the literature and the results showed that MVNS outperformed all of them in all of the cases tested. The work introduced by Selvi and Manimegalai [138] studied the use of a Two-Phase Variable Neighbourhood Search (TPVNS) for job scheduling on heterogeneous computing and grid systems. Six different neighbourhood structures were introduced and random PALS was also used. The performance of the proposed method was evaluated against a number of methods discussed in the literature and the results showed that TPVNS outperformed them in most of the cases investigated.

Xhafa *et al.* [171] proposed a new TS meta-heuristic for solving the problem of independent batch job scheduling in static and dynamic grid computing systems in terms of minimising the makespan and flowtime. The authors used the direct representation to encode the problem and a weighted sum function to combine the two objectives. The initial solution was generated using the min-min [77] *ad hoc* heuristic. Two types of movements were used, namely swap and transfer, and the different neighbours were explored according the load of resources in which jobs are moved from one high- to low-loaded resources to maintain a load balancing. The classical static dataset of Braun *et al.* was used to test the performance of the proposed TS and the experimental results showed that TS outperformed the results achieved by ACO+TS of Ritchie *et al.* [135] and cMA of Xhafa *et al.* for most

of the instances. Additionally, the authors evaluated the proposed TS using larger static problem instances in which the HyperSim-G grid simulator [27] was used to generate them and the experimental results showed that the proposed TS obtained the best results in terms of makespan and flowtime. Furthermore, the authors applied the proposed method to the dynamic version of the problem in which the rescheduling technique was used to introduce dynamism. Again, the proposed TS clearly outperformed the other algorithms explored in the study.

Nesmachnow *et al.* [117] considered a static independent batch job scheduling problem in heterogeneous environments in which the makespan of a schedule is to be minimised. The authors used an evolutionary algorithm, called parallel Cross generational elitist selection Heterogeneous recombination, and Cataclysmic mutation (pCHC) [46], to solve the scheduling problem in which a general-purpose tool for combinatorial optimisation problems, called MALLBA, was used to develop it. pCHC is a meta-heuristic method that could be considered as a variant of GA. It maintains diversity in the population through the use of an elitist selection operator. Additionally, a special reproducing operator is employed that applies a non-cluster role which guarantees producing new diverse individuals. pCHC uses the HUX operator to perform the crossover operation. Unlike a GA, mutation is not performed in pCHC. If a premature convergence is detected, a re-initialization procedure is performed in which the best solution achieved so far is used as a template for generating a new population. The proposed pCHC used the permutation-based representation to encode the problem. Two ad hoc heuristics, namely min-min and Sufferage, were used along with the random method to generate the initial population of pCHC. The authors used a cluster with four high speed servers to run their experiments. The classical static dataset of Braun *et al.* was used to test the performance of pCHC and the experimental results showed that it outperformed the results achieved by the other meta-heuristics [19] [135] [160] [165] [171] described in the literature in most of the instances. Moreover, the authors reviewed the existing benchmarks in the literature. They came to the conclusion that none of the available datasets can actually simulate the current characteristics of grid computing systems in terms of the dataset size. Therefore, they proposed a new benchmark, known as the Nesmachnow *et al.* dataset, to cover larger cases. Each case consists of 24 instances which were generated using the range-based method described in Ali *et al.* [8]. The proposed pCHC was also applied to the new benchmark and its results were compared with some heuristics from the bibliography in which it outperformed all of them. Later, Nesmachnow *et al.* [118] extended their work by proposing a new strongly coupled hybrid meta-heuristic, called micro pCHC, that combines pCHC with a local search procedure, called Problem Aware Local Search (PALS) that will be described in Chapter 4. Using PALS allowed the new hybrid method to achieve better results than those obtained by the stand-alone pCHC.

Several other meta-heuristic methods are also available in the literature. A fuzzy PSO meta-heuristic algorithm for task scheduling on computational grid was suggested by Liu *et al.* [93]. Their work focused on the minimisation of the makespan time. The authors used small- to large-scale resource-job pair problems to test the associated performance. Their method was compared with a GA and a SA approaches. The results showed that the fuzzy PSO scheduler has the ability to find faster and feasible solutions over the GA and SA. A differential evolution (DE) algorithm was developed by Selvi *et al.* [139] to generate schedules which efficiently utilise available resources and complete the jobs in the minimum time. The results have been compared with findings in [93] and it has been found that PSO outperforms DE in three instances. However, the authors in [139] claimed that the solutions found by DE show better resource utilisation.

3.6.2.4 Summary of meta-heuristics for job scheduling in grid computing

The problem of job scheduling in grid computing has been addressed using different approaches such as simple queuing algorithms, deterministic heuristic algorithms and meta-heuristic algorithms. However, to effectively deal with its complexity, meta-heuristic algorithms are preferred [85]. Meta-heuristic algorithms are well-known approaches which have been applied effectively to a wide range of NP-hard problems. In fact, these algorithms are considered the best candidate in practice to cope with the complexity of job scheduling in a computational grid, and accordingly several algorithms have been suggested [124].

One of these meta-heuristics is ACO, which has been shown to work well for various combinatorial static and dynamic problems such as railway junction scheduling [42] [43], data streams clustering [49] [50], TSP [111] [112] and vehicle routing problem [107] [108], which are closely related to the job scheduling problem in the heterogeneous environments such as grid computing. Based on the reviewed papers described in the literature, which are summarized in Table 3.2, several observations can be highlighted. One observation includes the use of ACO for job scheduling in grid computing in which the ACO meta-heuristic can be considered one of the best candidates to address this problem. The available works focused on developing new pheromone updating rules and/or on how to select the next job-resource pair rules. However, it can be seen that the performance of the existing ACO-based schedulers faced one problem which is the time ACO-based algorithms take before starting constructing good schedules as the pheromone trail is accumulated after performing a few iterations to identify the best job-resource mappings. This problem has been addressed in several ways.

Ritchie *et al.* [135] suggested an initial update of pheromone trials to avoid these iterations and speed up the convergence rate. To perform this update, they used the solution generated by the min-min heuristic in which the pheromone levels of the resources used

Table 3.2 Summary of ACO-based approaches for job scheduling problem in grid computing.

Author (s)	Algorithm (s)	Initialization	Hybridization type	Scheduling type
Ritchie *et al.* (2004)	ACO + TS	min-min	loosely coupled	static independent
Fidanova and Durchova (2006)	ACO	random	stand-alone	static independent
Lorpunmanee *et al.* (2007)	ACO	random	stand-alone	dynamic independent
Kousalya and Balasubramanie (2009)	ACO + 5 heuristics	random	loosely coupled	static independent
Mathiyalagan *et al.* (2010)	ACO	random	stand-alone	static independent
MadadyarAdeh and Bagherzadeh (2011)	ACO	random	stand-alone	static independent
Ku-Mahamud *et al.* (2012)	ACO	random	stand-alone	static independent
Christina and Miriam (2012)	ACO	random	stand-alone	static independent
Alobaedy and Ku-Mahamud (2014)	ACO + GA	random	loosely coupled	static independent

by the achieved schedule are increased. However, their proposed ACO approach took 3.5 hours to achieve the reported results. On the other hand, another solution to the slow convergence problem of ACO-based algorithms is to decrease the number of ants in the colony. Mavrovouniotis *et al.* in references [109] [110] suggested the reduction of the total of number of ants to 2 to allow the algorithm to perform more iterations and to reduce the time complexity. Fidanova and Durchova [54] used one ant, while Ku-Mahamud *et al.* [90] used a population of 7 ants. Additionally, the slow convergence problem of ACO-based algorithms can be solved, as suggested in a recent review on the use of ACO for job scheduling problem in grid computing conducted by Oshin and Chhabra [123], by hybridizing ACO with other meta-heuristics. The literature showed that for several NP-hard combinatorial optimisation problems, the best reported results were achieved when hybrid ACO algorithms were used [21] [36] [61] [103] [143] [144] [145] [146].

However, from the analysed works, it appears that only few studies have considered the hybridizing of ACO with other local search and meta-heuristic methods. More precisely, all the available hybrid ACO considered the loosely coupled fashion only and none of the available approaches, to the best of our knowledge, addressed the job scheduling problem in grid computing in a strongly coupled fashion. Additionally, despite the many ACO-based algorithms available in the literature which have tackled the static job scheduling problem in grid computing, only one research effort [94] examined ACO for the dynamic mode. Hence, there remains room to contribute to these lines of research.

On the other hand, the literature includes another meta-heuristic algorithm, which is the GA, that has been progressively used for solving the various scheduling problems in grid computing and other similar environments since the early 1990s. The analysis of the available proposals, which are summarized in Table 3.3, allows identifying several observations.

The first observation includes the large use of GAs and their variants to tackle many types of job scheduling problems in grid computing systems and distributed heterogeneous environments, which makes GAs a promising technique to solve scheduling problems. The early proposals, from 1996 to 2000, focused on formulating the job scheduling problem. The main aim of applying the proposed GA-based schedulers was to study the behaviour of evolutionary techniques in such complex problems. In general, most of proposals in this period considered the static dependent version of the problem and small-scale datasets were used mainly due to the limited multi-processor architecture at that time. Although the used datasets were small, the reported results achieved by the stand-alone GA-based algorithms showed that the proposed approaches required more processing time to obtain good quality schedules. Therefore, researchers have often hybridized GAs with other local search and heuristic methods as an attempt to resolve this problem.

In early 2000s, based on the work conducted by Ali *et al.* [8], who proposed two methods for generating problem instances and benchmarks for heterogeneous environments such as grid computing, Braun *et al.* [19] presented a clearer formulation of the static independent job scheduling problem which aims to minimise the makespan. The adoption of independent job scheduling problems led to accurately simulate more realistic cases of distributed autonomous heterogeneous environments such as grid environments. Braun *et al.* generated what is known now as the *de facto* static benchmark for studying and evaluating different heuristic and meta-heuristic job scheduling proposals in distributed heterogeneous environments. Moreover, they applied GAs to solve the problem and compared their performance to many other heuristics, in which GAs showed their efficiency in tackling such a complex problem.

In the period 2001-2005, an important observation can be noted, which is the addressing of dynamic scheduling problems. Although, this type of scheduling is more complex than the static version, GAs proved their efficiency in this field compared to other methods. Additionally, the use of various *ad hoc* heuristics to seed the initial population of GAs was very clear. Furthermore, this period also witnessed the use of several strongly coupled hybrid GA-based schedulers which showed consistently their efficiency over other stand-alone algorithms.

Another important period is from 2006 to 2014, in which Xhafa *et al.* proposed and evaluated various GA variants to tackle the job scheduling problem from different perspectives. Their research efforts provided the necessary baselines and references to the literature and greatly influenced the researchers, including the author, to undergo

Table 3.3 Summary of GA-based approaches for job scheduling problem in grid computing.

Author (s)	Algorithm (s)	Initialization	Hybridization type	Scheduling type
Tirat-Gefen and Parker (1996)	GA + LP	random	stand-alone	static dependent
Shroff *et al.* (1996)	GSA	random	strongly coupled	static dependent
Wang *et al.* (1997)	GA	baseline	stand-alone	static dependent
Abraham *et al.* (2000)	GA + SA, GA + TS	LJFR–SJFR	strongly coupled	static independent
Braun *et al.* (2001)	GA	min-min	stand-alone	static independent
Theys *et al.* (2001)	GA	min-min	stand-alone	static independent, static and dynamic dependent
Grajcar (1999; 2001)	GA + LS	random	strongly coupled	static dependent
Zomaya and The (2001)	GA	sliding-window	stand-alone	dynamic independent
Page and Naughton (2005)	GA	random	stand-alone	dynamic independent
Prodan and Fahringer (2005)	GA	job migration, cycle elimination	stand-alone	static and dynamic dependent
Sugavanam (2005; 2007)	Max-Max, GIM, OIM,GENITOR, MA, ACO, HEA	Max-Max	stand-alone strongly coupled	static dependent
Carretero and Xhafa (2006; 2007)	GA	MCT, LJFR–SJFR	stand-alone	static independent
Xhafa and Duran (2008)	Struggle GA	MCT, LJFR–SJFR	stand-alone	static independent
Xhafa *et al.* (2008)	cMA	LJFR-SJFR	loosely coupled	static independent
Xhafa and Carretero (2009)	GA	MCT, LJFR–SJFR	stand-alone	static and dynamic independent
Xhafa *et al.* (2009)	GA + TS	MCT, LJFR-SJFR	strongly coupled	static and dynamic independent
Xhafa *et al.* (2011)	GA + TS	MCT, LJFR-SJFR	loosely coupled	static and dynamic independent
Xhafa *et al.* (2007; 2011)	MA + TS	MCT, LJFR-SJFR	loosely coupled	static and dynamic independent
Falzon and Li (2012)	GA	several heuristics	strongly coupled	static dependent
Kardani-Moghaddam *et al.* (2012)	GA + VNS	MinCTT, SuffCTT	strongly coupled	static independent
Kolodziej *et al.* (2011)	Struggle GA	MCT, LJFR-SJFR	stand-alone	static and dynamic independent
Kolodziej *et al.* (2012)	GA	MCT, LJFR-SJFR	stand-alone	static and dynamic independent
Kolodziej *et al.* (2014)	GA	MCT, LJFR-SJFR	stand-alone	static and dynamic independent
Zahedani and Dastghaibyfard (2014)	GA + PSO	random	strongly coupled	static and dynamic independent
Oshin and Bhatt (2017)	PSO + CS + GA	random	loosely coupled	static independent

future investigations about diverse job scheduling problems. These research efforts include analysing many GA variants, such as Steady-State and Struggle GAs, MAs and cMAs, and hybrid GAs, in static and dynamic modes. Several datasets were used to evaluate their various proposals. In addition to the Braun *et al.* dataset to which they obtained one of the best-achieved results, Xhafa *et al.* generated several random problem instances for static and dynamic modes using the HyberSim-G grid simulator [27]. However, these instances are not available since they were generated randomly. In most of the works presented, Xhafa *et al.* optimized the makespan and flowtime using a weighted sum function. However, more objectives that are related to energy and security were added to them. Furthermore, as they were optimizing the makespan and flowtime simultaneously, the use of MCT and LJFR−SJFR heuristics appeared in most of their research which is due to the fact that these *ad hoc* heuristics provide the best start up solutions in terms of these two objectives. Another key issue that was investigated by Xhafa *et al.* is the hybridisation of GAs with other approaches in which several local search and meta-heuristics were proposed that were mainly depended on the concepts of random move and swap. The results achieved by the hybrid methods, for the static and dynamic modes, outperformed the other stand-alone approaches as showed in the corresponding works such as [167] and [170].

Besides ACO and GA, the literature includes other meta-heuristic methods that were frequently used to tackle the various job scheduling problems. A summary of the most relevant meta-heuristic techniques is presented in Table 3.4. Liu *et al.* [93] examined the use of PSO to solve the static independent job scheduling problem in which it achieved results that are better than GA and SA. However, similar to ACO-based schedulers, it suffered from the need for more time to build these results. The TS meta-heuristic was also applied to the job scheduling problem. Xhafa *et al.* [171] proposed a TS that reported one of the best known results for the Braun *et al.* dataset. The proposed TS used different types of movements that are based on the move and swap operations. Table 3.4 also includes another meta-heuristic, which is pCHC, that was applied to the job scheduling problem. Although they provided one of the best results for the Braun et al. dataset, Nesmachnow *et al.* [117] [118] used a very high speed network to obtain these results.

Additionally, the literature includes another meta-heuristics, which is VNS, that is based on the systematic change of the neighbourhood during the search process. Its ability to explore different neighbours allows VNS to search complex large state spaces. As can be seen in Table 3.4, three works tackled the static dependent and independent job scheduling using VNS meta-heuristic, i.e., it has not been previously used, to the best of the author's knowledge, for the dynamic mode. Davidovic *et al.* used a simple one neighbourhood structure that is based on a random swap operation. On the other hand, recently, Selvi *et al.* [137] [138] used more structures which were developed based on the random swap and move concepts.

Table 3.4 Summary of other meta-heuristics approaches for job scheduling problem in grid computing.

Author (s)	Algorithm (s)	Initialization	Hybridization type	Scheduling type
Davidovic *et al.* (2001)	VNS, TS	CP, LS	stand-alone	static dependent
Liu *et al.* (2010)	PSO, GA, SA	LJFR-SJFR	stand-alone	static independent
Selvi *et al.* (2011)	DE	random	stand-alone	static independent
Xhafa *et al.* (2012)	TS	min-min	stand-alone	static and dynamic independent
Nesmachnow *et al.* (2012)	pCHC	min-min, Sufferage	stand-alone	static independent
Nesmachnow *et al.* (2012)	pCHC + PALS	min-min, Sufferage	loosely coupled	static independent
Selvi and Manimegalai (2015)	VNS	min-min	stand-alone	static independent
Selvi and Manimegalai (2015)	GVNS+BVNS	min-min	loosely coupled	static independent

Despite the many hybrid meta-heuristic algorithms available in the literature which have tackled the job scheduling problem in grid computing, only one research effort [170] compared between the use of loosely and strongly coupled hybrid meta-heuristics in which a random dataset was used. Moreover, although VNS, whether it has been hybridized with other algorithms or employed as a stand-alone algorithm, has proved its effectiveness in dealing with numerous complex hard optimisation problems [72], only one research effort [83] has previously studied the hybridization of the VNS algorithm with other meta-heuristics with regard to the static independent job scheduling problem on computational grids. Hence, there remains room to contribute to these lines of research.

3.7 Summary

This chapter introduced the main concepts behind grid computing and its main components. One of the main challenges in a computational grid is how to efficiently map jobs, also called tasks or applications, to grid resources and hence utilize geographically distributed computers which are connected through heterogeneous environments in an efficient, reliable and secure manner. This mapping is called job scheduling in grid computing. The mathematical formulation of the job scheduling problem in grid computing was described in this chapter. Moreover, the main methods by which benchmarks can be generated were also presented. Similar to job scheduling in traditional computing systems, this mapping is known to be an NP-hard problem [161]. However, it is more complicated in grid computing due to its complex, dynamic nature, high degree of job and resource heterogeneity, problem size, and other factors such as existing local schedulers and policies

[25]. Therefore, exact methods are not applicable since they required exponential time to achieve such mappings. Alternatively, heuristics and meta-heuristics provide near-optimal solutions in a reasonable time. The literature involves several heuristic and meta-heuristics that were applied to address the problem. This chapter presented a comprehensive review of the static and dynamic heuristic and meta-heuristic approaches that were used to tackle the job scheduling problem in heterogeneous environments such as grid computing.

Chapter 4

Hybrid Meta-Heuristics for Static Job Scheduling in Grid Computing

Several stand-alone meta-heuristics, such as Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Variable Neighborhood Search (VNS) and Tabu Search (TS), have been applied successfully for various types of scheduling problems. However, the results achieved by these methods could be further improved by combining two or more meta-heuristics [10]. The resulting new high-level algorithm would then inherit the best features of the combined meta-heuristics. Consequently, the chances of escaping from a local minimum will be increased, and hence the overall performance will be enhanced [170].

In general, there are two means, or fashions by which to effectively combine meta-heuristics, namely loosely coupled and strongly coupled [167]. The first fashion consists of executing the combined meta-heuristics in a serial manner such that the solution to the first method then being used by the second and so on; the final solution will be the output of the last algorithm. The second fashion refers to the type of hybridization in which the inner procedures of the hybridized algorithms are interchanged in such a way that one of the methods acts as the main algorithm, which during its execution calls other methods to act as supporting algorithms [169].

In this chapter, the use of VNS for job scheduling in grid computing is introduced. Four new neighborhood structures, together with a modified local search, are proposed. The proposed VNS is hybridized with two meta-heuristic methods in a loosely and strongly coupled fashions, yielding new hybrid meta-heuristic algorithms by which to consider the job scheduling problem in grid computing.

4.1 The solution representation

A key issue in meta-heuristic algorithms is the representation of solutions. Two types of encodings have been reported in the literature for the job scheduling problem in grid computing: the job-based representation, also called the direct representation, and the resource-based representation, also known as the permutation-based representation [25]. For all the proposed meta-heuristic algorithms, we will consider both representations.

Each solution is represented as a list in the job-based representation, as shown in Fig. 4.1. The list size is equal to the total number of jobs. The solution[a] represents the resource into which the job a is assigned. Hence, we should expect integers in the range [0, total number of resources-1] for this list.

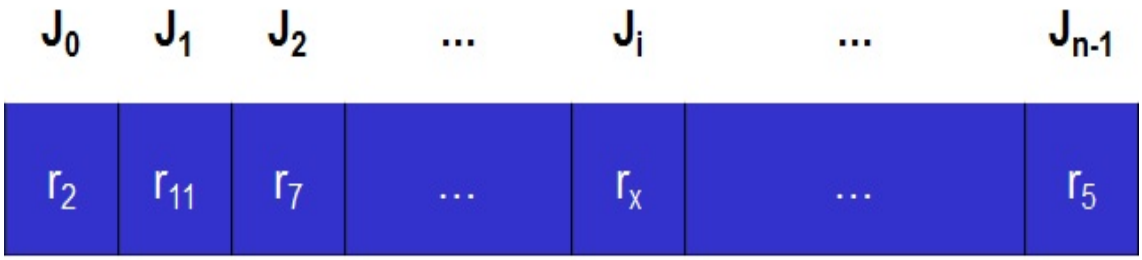

Fig. 4.1 An example of job-based representation for the job scheduling problem in grid computing.

In the resource-based scheme, each solution is also represented as a list, as shown in Fig. 4.2. However, the list size is equal to the total number of resources. The solution[a] represents the resource where a list of jobs will be assigned. Hence, we should expect integers in these lists in the range [0, total number of jobs-1]. Unlike the job-based representation where elements hold the resources which can be repeated, in this representation each element represents a list of jobs which have been assigned to it and are unique.

4.2 The application of VNS to the job scheduling problem

VNS is a simple and effective meta-heuristic algorithm that is often applied to many optimisation problems; Mladenovic and Hansen [114] proposed VNS in 1997 as a flexible framework which can be used to define heuristics that are applicable to various problems. VNS uses multiple neighborhood structures to explore a number of neighborhoods for the current incumbent solution. It then picks the neighbor that introduces an improvement. The systematic change of neighborhood structures is the core concept of the VNS meta-heuristic. This change takes place in the descent phase where the meta-heuristic seeks to

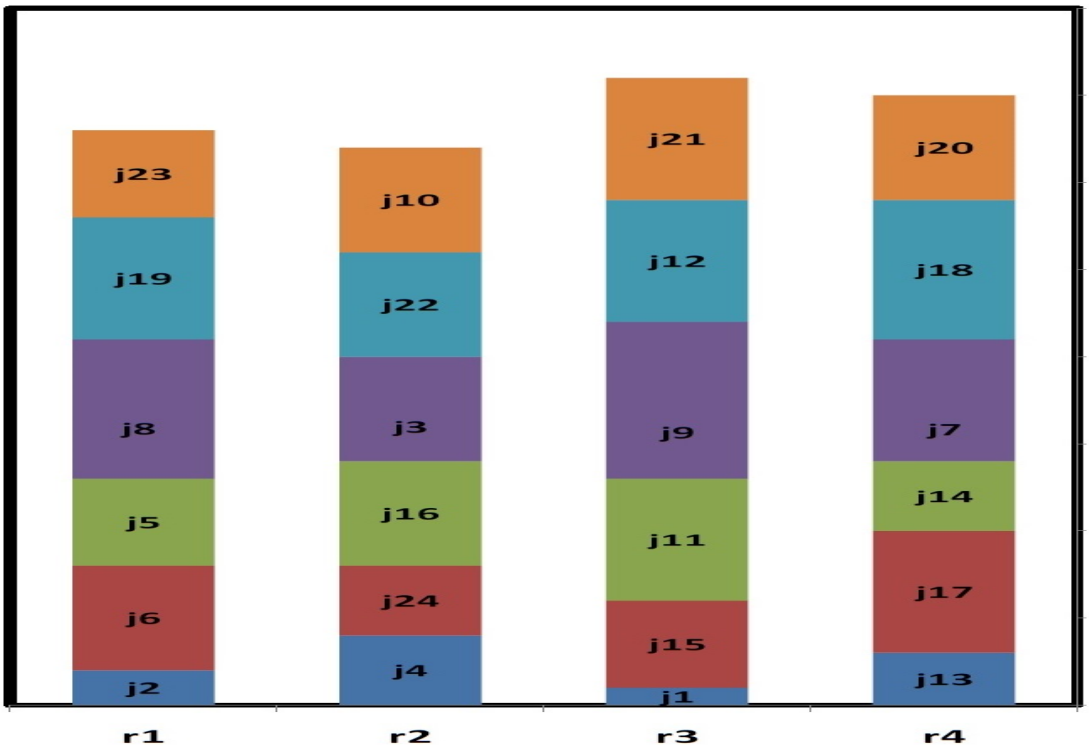

Fig. 4.2 An example of resource-based representation for the job scheduling problem in grid computing.

find a local optimum; it also occurs in the perturbation phase where the VNS attempts to escape from the local optimum.

Generally, the proposed VNS procedure for the job scheduling in grid computing (illustrated in Algorithm 4.1) involves three steps, which are repeated until the termination conditions are satisfied. These steps are:

1. The shake step.
2. The improvement step.
3. The neighbourhood change step.

4.2.1 Neighbourhood structures for job scheduling in grid computing

The first step, known as the shake step, includes the application of a set of operators in a particular order. The neighbourhood structure provides a way to explore new parts in the solution space. This exploration is achieved through defining the type of modifications which could be applied to a given solution to produce new ones. The solution space can

Algorithm 4.1: The proposed VNS procedure for the job scheduling problem in grid computing.

```
1  Let: S_0 ←min-min();
2  Let: N_k ←the set of neighbourhood structures, k ∈ [1, k_max];
3  repeat
4      k ← 1;
5      repeat
6          S_1 ← shake(S_0);
7          S_2 ← PALS(S_1);
8          if (fitness(S_2) < fitness(S_0)) then
9              S_0 ← S_2;
10             k ← 1;
11         else
12             k ← k + 1;
13         end
14     until (k==k_max);
15 until (termination condition);
```

be explored in different ways using different neighbourhoods; thus, using well-defined neighbourhood structures will certainly lead to better exploration.

In the job scheduling problem, any solution S will have at least one resource with a local makespan time equal to the overall makespan of the solution, where this resource is called the 'problem resource'. New solutions can be obtained from S by swapping a job currently assigned to the problem resource with a job assigned to other resource, or by moving a job currently assigned to the problem resource to a different resource. Therefore, we can define many new neighbourhood structures based on the concepts of swap and move. In this study, four new structures have been proposed, which are the Penalty-based Swap (PS), the Penalty-based Move (PM), the Random Max to Min Move (RMMM) and the Longest Max to Min Move (LMMTM).

The first structure, PS, alters the solution by finding the set of exchanges of some of the jobs assigned to the problem resource with some of the jobs assigned to other resources which best improve the solution in terms of minimising the makespan. The general PS procedure is illustrated in Algorithm 4.2. On the other hand, PM modifies the solution by finding the set of moves which reallocates some of the jobs assigned to the problem resource to other resources that best reduce the makespan time, as shown in Algorithm 4.3, while RMMM, the third neighbourhood structure, moves a random job from the list of the jobs assigned to the problem resource to the resource with the minimum local makespan time, as demonstrated in Algorithm 4.4. Finally, LMMM defines new neighbours by moving the job in the list of jobs assigned to the problem resource which has the maximum

Algorithm 4.2: The Penalty-based Swap (PS) procedure

```
1  Ś ← S;
2  Min_penalty ← makespan(S);
3  penalty ← 0;
4  Find the problem resource (pr) which has the maximum local makespan time;
5  Pr_Job_List ← job list of pr;
6  Pr_size ← Pr_Job_List size;
7  for (i=0 to Pr_size − 1) do
8      for (j=0 to total number of jobs-1) do
9          if (Ś[j] ≠ pr) then
10             Ś ← swap_resources(Ś, Pr_Job_List[i], j);
11         end
12         penalty ← makespan(Ś);
13         if (penalty < Min_penalty) then
14             S ← Ś;
15             Min_penalty ← penalty;
16         else
17             Ś ← S;
18         end
19     end
20 end
```

expected completion time to the resource which has the minimum processing time for it. Algorithm 4.5 describes the general steps of LMMM.

4.2.2 The improvement step

This step involves applying a local search procedure to improve the solution generated form the shaking step. For the job scheduling problem in grid computing, a modified version of the Problem Aware Local Search (PALS) is used. PALS was originally proposed to solve the problem of DNA fragment assembly problem [6] [7], and a variant thereof called Randomized PALS has been used for job scheduling in heterogeneous environments [117] [138]. Recently, it has been used as an efficient technique for some permutation-based optimisation problems [113].

The general steps of the modified random PALS are described in Algorithm 4.6. The algorithm selects a resource Pr, which is the resource with the largest local makespan, and a random resource Rres such that $Pr \neq Rres$ from a given solution S.

The outer loop iterates on some of the jobs of Pr. The number of these jobs is selected according to the generation of a random number, Pr_start, which belongs to the range [1, $Pr\_list\_size - 1$] where Pr_list_size is the number of jobs assigned to Pr and another random number, Pr_end, from the range [Pr_start, Pr_list_size].

Algorithm 4.3: The Penalty-based Move (PM) procedure

```
1  Ś ← S;
2  Min_penalty ← makespan(S);
3  penalty ← 0;
4  Find the problem resource (pr) which has the maximum local makespan time;
5  Pr_Job_List ← job list of pr;
6  Pr_size ← Pr_Job_List size;
7  for (i=0 to Pr_size − 1) do
8      for (j=0 to total number of jobs-1) do
9          if (Ś[j] ≠ pr) then
10             Ś ← move_job(Ś, Pr_Job_List[i],resource assigned to j);
11         end
12         penalty ← makespan(Ś);
13         if (penalty < Min_penalty) then
14             S ← Ś;
15             Min_penalty ← penalty;
16         else
17             Ś ← S;
18         end
19     end
20 end
```

Algorithm 4.4: The Random Max to Min Move (RMMM) procedure

```
1  Find the problem resource (pr) which has the maximum local makespan time;
2  Pr_Job_List ← job list of pr;
3  Pr_size ← Pr_Job_List size;
4  Find the resource (mr) which has the minimum local makespan time;
5  i ← Random number in the range [0..Pr_size];
6  S ← move_job(S, Pr_Job_List[i], mr);
```

Algorithm 4.5: The Longest Max to Min Move (LMMM) procedure

```
1  Find the problem resource (pr) which has the maximum local makespan time;
2  Pr_Job_List ← job list of pr;
3  Pr_size ← Pr_Job_List size;
4  Find the job (lj) in Pr_Job_List which has the longest processing time;
5  Find the resource (br) which is the fastest resource that can process lj;
6  S ← move_job(S, lj, br);
```

Algorithm 4.6: The Modified Random Problem Aware Local Search Procedure

```
1  for (iter=1 to max_iter) do
2      sol_makespan ← makespane(S);
3      Ś ← S;
4      Min_makespan ← ∞;
5      Find the problem resource (Pr) which has the maximum local makespan time;
6      Pr_Job_List ← job list of the resource with maximum local makespan;
7      Pr_size ← Pr_Job_List size;
8      Randomly select a resource Rres such that Rres ≠ Pr;
9      Rres_Job_List ← job list of the resource with minimum local makespan;
10     Rres_size ← Rres_Job_List size;
11     Pr_start ← random(1, Pr_size − 1);
12     Pr_end ← random(Pr_start, Pr_size);
13     Rres_start ← random(1, Rres_size − 1);
14     Rres_end ← random(Rres_start, Rres_size);
15     for (i=Pr_start to Pr_end) do
16         for (j=Rres_start to Rres_end) do
17             S1 ← swap_resources(Ś, Pr_Job_List[i], Rres_Job_List[j]);
18             S2 ← move_job(Ś, Pr_Job_List[i], Rres);
19             if (makespan(S1) < makespan(S2)) then
20                 S̈ ← S1;
21             else
22                 S̈ ← S2;
23             end
24             current_makespan ← makespan(S̈);
25             if (current_makespan < Min_makespan) then
26                 Ś ← S̈;
27                 Min_makespan ← current_makespan;
28             end
29         end
30     end
31     if (Min_makespan < sol_makespan) then
32         S ← Ś;
33     end
34 end
```

Similarly, the inner loop works on the jobs of Rres using *Rres_start* and *Rres_end* which are generated in the same manner as *Pr_start* and *Pr_end*, respectively. This process, which was suggested in [138], guarantees the selection of different jobs with different sizes in each iteration, while the randomised PALS used in [117] has used a different method in which the start job of each loop is generated randomly and the number of jobs in the outer loop is fixed to 32, whereas the number of jobs in the inner is set to the number of jobs divided by 20.

The double loop calculates the makespan values when swapping and moving jobs in S1 and S2, respectively. Then it selects the solution with the minimum makespan and compares this with the best solution so far; if there is an improvement, then it will become the new best; otherwise, it will continue. Therefore, this loop stores the best improvement to the solution with respect to the makespan time obtained by applying *Pr_end X Rres_end* swaps or transfers.

This solution is then compared with S to decide whether to accept or reject it. If it is better than S, then the algorithm will accept it as the new S; otherwise, it will reject it and continue. The process is repeated max_iter times which means that the best improvement is applied, whereas the random PALS used in [117] and [138] is applied until an improvement to the original solution is discovered or until max_iter iterations, i.e., the first improvement strategy was used.

4.3 The application of hybrid ACO to the job scheduling problem

ACO is a meta-heuristic search algorithm which simulates the behaviour of ants in the process of foraging for food and how they can find a path between their nest and a source of food in this process [38]. ACO has been successfully applied for many NP complete problems which are closely related to the problem of job scheduling in computational grids [41]. This section introduces two hybrid ACO-based schedulers for the job scheduling problem in grid computing.

Selecting a structure to represent the solution of the problem under examination is the first step in any ACO-based algorithm. A colony of k ants is used in which each ant represents the whole solution to the scheduling problem.

The ACO algorithm uses two types of information to find a solution to an optimisation problem, namely the pheromone trial and the heuristic information. The ants use the pheromone trail to communicate between them. This communication involves sharing useful information about optimal solutions. A pheromone matrix, τ, of size n x m is required, where τ[j][r] represents the favourability of allocating job j to resource r. The second piece of information the ants use to construct their solutions is the heuristic function η_{jr}. The following heuristic is used, which was proposed in [54]:

$$\eta_{jr} = \frac{1}{free[r]} \tag{4.1}$$

where the function free[r] represents the time by which the resource r becomes free. η_{jr} will be a large value if free[r] is small. Thus, a resource will be more desirable if it is free earlier.

A fitness function is required to measure the quality of the solutions, which is the makespan in our case. The general throughput of the grid system can be indicated by the makespan value of the schedule; a small makespan means that the scheduler is producing a high-quality mapping that best utilizes the available resources.

After each iteration, the pheromone deposit is updated as defined in Equation 4.2, which follows the updating rule described in [146]. This update allows the indirect communication among ants to share information about the current states of the resources.

$$\tau_{jr} = \begin{cases} \rho * \tau_{jr} + \Delta\tau_{jr} & \text{if job j is assigned to resource r in local\_best\_ant} \\ \rho * \tau_{jr} & \text{otherwise} \end{cases} \tag{4.2}$$

where ρ, $(0 < \rho \leq 1)$, represents the decay parameter the ants use to forget poor solutions and $\Delta\tau_{jr}$ denotes the amount of pheromone deposit on the path and is defined by Equation 4.3 as:

$$\Delta\tau_{jr} = \frac{makespan(local\_best\_ant)}{makespan(global\_best\_ant)} \tag{4.3}$$

Rules 4.2 and 4.3 allow the best ant only to deposit pheromone after each iteration. The best ant could be defined as the *local_best_ant* (the best ant in the current iteration) or the *global_best_ant* (the best ant so far).

To construct its solution, each ant uses the heuristic function as well as the information encoded in the pheromone trial. Moreover, every ant maintains two lists, namely mapped and unmapped lists. The former is initially empty, while the latter contains all the submitted jobs. The ACO-based scheduler chooses the first job-resource pair randomly. The next job-resource pair is picked out probabilistically by mapping job j to resource r using the transition rule defined in Equation 4.4 as follows:

$$p_{jr} = \frac{[\tau_{jr}]^{\alpha} * [\eta_{jr}]^{\beta} * \frac{1}{ETC[j,r]}}{\sum[\tau_{jr}]^{\alpha} * [\eta_{jr}]^{\beta} * \frac{1}{ETC[j,r]}} \tag{4.4}$$

where τ_{jr} and η_{jr} are the existing pheromone trail and the heuristic information, respectively, and α and β are two parameters used to define the relative weights of the pheromone and the heuristic, respectively. The pheromone trail τ_{jr} provides each ant with information about the favourability of assigning job j to resource r. On the other hand, the heuristic function η_{jr} will find the best available resource r, in terms of being free earlier, to process job j from the unmapped jobs list. Furthermore, the inverse of ETC[j,r] is used as an additional heuristic function; the inverse was used since lower values are more preferable.

The same steps are repeated until the unmapped list becomes empty, i.e., a solution is constructed. The same procedure is followed by every ant in the colony. When all k ants

construct their solutions, the best local ant in the colony is determined. The pheromone update rule, defined in Equation 4.2, is then applied.

It is worth mentioning that the pheromone trail update rule is also applied at the beginning before starting the main ACO procedure, where the solution found by the deterministic heuristic min-min algorithm [77], described in Chapter 3, is used to update the pheromone trail in order to speed up the process of finding good solutions.

4.3.1 Hybridizing ACO with VNS for the job scheduling problem

The ACO scheduler can be hybridized with VNS either in a loosely or strongly coupled fashion, yielding two new hybrid meta-heuristic algorithms by which to consider the job scheduling problem in grid computing. The loosely coupled hybrid algorithm, called ACO+VNS, combines ACO and VNS, in which the former works first and whose output is further refined by the latter algorithm as described in Algorithm 4.7. On the other hand, the strongly coupled hybrid algorithm, called ACO(VNS), also combines ACO and VNS, in which the former acts as the main algorithm, which during its execution calls the latter algorithm to act as a supporting algorithm. When all k ants construct their solutions, the best local ant in the colony is determined and the VNS algorithm is performed on it before updating the pheromone trail. The VNS improves the solution found by the local best ant and hence there is a good possibility that the local best ant will be the next global best ant. The main ACO(VNS) steps are illustrated in Algorithm 4.8.

4.4 The application of hybrid GA to the job scheduling problem

This section introduces the use of hybrid GAs for the job scheduling problem in grid computing. The following subsections explain the main parts of each hybrid GA.

4.4.1 The initial generation

The random method is a common way to construct the initial generation of the genetic algorithm. However, several studies have showed that seeding the initial population of a genetic algorithm with solutions from other heuristic methods will introduce greater diversity and hence produce better solutions [175]. In this study, the initial population is generated as follows: one individual will be seeded with the solution found by the *ad hoc* min-min heuristic algorithm [77], which is described in Section 5. The remaining solutions are generated randomly.

Algorithm 4.7: The ACO+VNS scheduler procedure.

let *num* and *res* be the total number of jobs and resources respectively;
Set the pheromone trial τ_{nm} to a small value;
Initialise $free[0..res-1]$ to 0;
Initialise the pheromone evaporation ρ;
Initialise *global_best_ant* to the solution found by *min_min* algorithm;
ms $\leftarrow$ the makespan of *global_best_ant*;
$\Delta\tau_{nm} \leftarrow \frac{1}{ms}$;
Use Equation 4.2 to update the pheromone trail;
while *(the stopping condition is not true)* **do**
    for *(every ant)* **do**
        Randomly select the job-resource pair (a, b);
        Add (a, b) to the mapped list;
        for *(all unmapped jobs)* **do**
            $free[b] \leftarrow free[b] + ETC[a,b]$;
            Use Equation 4.1 to compute the heuristic function;
            Use Equation 4.4 to compute the probability matrix;
            Find the highest ρ_{wv} value;
            Determine the next job-resource pair (a=w, b=v);
            Append (a, b) to the mapped list;
        end
    end
    Compute the makespan of every ant;
    Find *local_best_ant*, which is the one with the minimum makespan;
    if *((makespan(local_best_ant) < makespan(global_best_ant))* **then**
        global_best_ant $\leftarrow$ *local_best_ant*;
    end
    Use Equation 4.3 to compute $\Delta\tau_{nm}$;
    Use Equation 4.2 to update the pheromone trail;
end
Apply VNS algorithm, i.e., *global_best_ant* $\leftarrow$ VNS(*global_best_ant*);

4.4.2 The fitness evaluation

As mentioned earlier, this work will focus on minimising the makespan. Therefore, the fitness of solutions is evaluated using Equation 3.1. See Chapter 3, Section 3.4 (Job scheduling in grid computing problem formulation) for details.

4.4.3 The selection operator

Several selection techniques are available in the literature. In this study, the N-Tournament method suggested in [25] is used with N=4. In the tournament selection, several tournaments are run among a few individuals which have been selected randomly from the

Algorithm 4.8: The ACO(VNS) scheduler procedure.

```
1  let num and res be the total number of jobs and resources respectively;
2  Set the pheromone trial τ_nm to a small value;
3  Initialise free[0..res − 1] to 0;
4  Initialise the pheromone evaporation ρ;
5  Initialise global_best_ant to the solution found by min_min algorithm;
6  ms ← the makespan of global_best_ant;
7  Δτ_nm ← 1/ms;
8  Use Equation 4.2 to update the pheromone trail;
9  while (the stopping condition is not true) do
10     for (every ant) do
11         Randomly select the job-resource pair (a, b);
12         Add (a, b) to the mapped list;
13         for (all unmapped jobs) do
14             free[b] ← free[b] + ETC[a,b];
15             Use Equation 4.1 to compute the heuristic function;
16             Use Equation 4.4 to compute the probability matrix;
17             Find the highest ρ_wv value;
18             Determine the next job-resource pair (a=w, b=v);
19             Append (a, b) to the mapped list;
20         end
21     end
22     Compute the makespan of every ant;
23     Find local_best_ant, which is the one with the minimum makespan;
24     Apply VNS algorithm, i.e., local_best_ant ← VNS(local_best_ant);
25     if ((makespan(local_best_ant) < makespan(global_best_ant)) then
26         global_best_ant ← local_best_ant;
27     end
28     Use Equation 4.3 to compute Δτ_nm;
29     Use Equation 4.2 to update the pheromone trail;
30 end
```

population. The winner of each tournament (the one with the best fitness) is selected for the next stage.

4.4.4 The crossover operator

The evolutionary computing literature contains several types of crossover operators, which mainly depend on the solution encoding method. The six crossover operators, which were described in Section 2.5.5.1, will be considered. More precisely, three crossover operators, which are the one-point crossover (1P), two-point crossover (2P) and half uniform crossover (HUX), will be used for the direct representation, while the Order

Crossover (OX), Partially Matched Crossover (PMX) and Cycle Crossover (CX) operators will be examined for the resource-based representation.

4.4.5 The mutation operator

In this work, several mutation operators have been used, each of which is applied on each individual with a probability of p_m. These operators are:

1. Random move: A random job, J_x, from a random resource, R_z, is selected, then it is assigned to a random resource, R_w, such that $R_z \neq R_w$.

2. Random swap: Two random jobs, J_x and J_y, are selected and their corresponding resources, R_y and R_z respectively, are exchanged.

3. Best move: This operators alters the solution by transferring the job assigned to the problem resource which has the maximum expected completion time to the resource which has the minimum processing time for it. Algorithm 4.9 demonstrates the pseudo-code for the best transfer mutation.

4. Best swap: This operator alters the solution by finding the best resource swap between one of jobs assigned to the problem resource and all other jobs which best minimise the makespan. The pseudo-code for the best swap mutation is illustrated in Algorithm 4.10.

Algorithm 4.9: The best move mutation

```
1 Find the problem resource (pr) which has the maximum local makespan time;
2 Find the list of jobs assigned to pr (pr_list);
3 Find the job j in pr_list which has the maximum expected completion time;
4 Assign j to resource r which has the minimum processing time for j;
```

4.4.6 The replacement operator

In this thesis, the Steady State Strategy is used, that is, parents and offspring compete for survival, and then the best of them are selected. Although this causes a premature stagnation of the population, the use of the Steady State Strategy produces a fast convergence (minimization) of the objective function [163] [25]. This characteristic serves our goals since we are interested in minimising the makespan in a relatively short time.

Algorithm 4.10: The best swap mutation

```
1  Find the problem resource (pr) which has the maximum local makespan time;
2  Find the list of jobs assigned to pr (pr_list);
3  for All pj ∈ pr_list do
4      for All oj ∈ all jobs and oj ≠ pj do
5          new_solution ← swap the resource assigned to pj with the resource
             assigned to oj;
6          calculate the fitness of new_solution;
7          add new_solution to the list of all solutions;
8      end
9  end
10 Find the swap move in the list of all solutions which best minimise the overall
     makespan;
```

4.4.7 Hybridizing GA with VNS for the job scheduling problem

The GA algorithm can be hybridized with VNS either in a loosely or a strongly coupled fashion, yielding two new hybrid meta-heuristic algorithms by which to consider the job scheduling problem in grid computing. The loosely coupled hybrid algorithm, called GA+VNS, combines GA and VNS, in which the former works first and whose output is further refined by the latter algorithm as described in Algorithm 4.11. On the other hand, the strongly coupled hybrid algorithm, called GA(VNS), also combines GA and VNS, in which the former acts as the main algorithm, which during its execution calls the latter algorithm to act as a supporting algorithm. For this purpose, the proposed VNS procedure, described in Algorithm 4.1, can be employed as a mutation operator. The main GA(VNS) steps are illustrated in Algorithm 4.12.

4.5 Summary

In this chapter, the application of three meta-heuristic methods, VNS, ACO and GA, for the independent job scheduling problem in grid computing has been introduced. ACO and GA have been hybridized with a novel VNS in both loosely and strongly coupled fashions. The new high-level algorithms inherit the best characteristics of the combined methods. Four new neighbourhood structures and a modified PALS have been proposed for the novel VNS, which use the concepts of move and transfer of some jobs to or from the problem resource, which is the resource that has a local makespan equal to the total makespan of the solution. Through the use of these structures and the modified local search, VNS improves the performance of the ACO and GA algorithms by introducing diversity to the colony and the population, respectively, and by exploring new parts of the state space of the problem.

Algorithm 4.11: The hybrid GA-VNS algorithm

```
1  t ← 0;
2  Generate the initial generation Gen(t) of k individuals, where Gen(t)[0]=
     min_min() and the remaining individuals,Gen(t)[1] to Gen(t)[k-1], are generated
     randomly;
3  Evaluate the fitness of each individual in the initial generation, i.e., compute
     Fitness(Gen(t));
4  while (the end criterion is not true) do
5  |   t ← t + 1;
6  |   Select Parent(t) from Gen(t-1);
7  |   With probability p_c, recombine individuals in Parent(t) to produce Offspr1(t);
8  |   With probability p_m, mutate individuals in Offspr1(t) to produce Offspr2(t);
9  |   Evaluate the fitness of each individual, i.e., compute Fitness(Offspr2(t));
10 |   Replace Gen(t) from Offspr2(t) and/or Gen(t-1);
11 end
12 Find best_S which is the best solution in Gen(t);
13 Apply VNS algorithm, i.e., best_S gets VNS(best_S) return best_S;
```

Algorithm 4.12: The hybrid GA(VNS) algorithm

```
1  t ← 0;
2  Generate the initial generation Gen(t) of k individuals, where Gen(t)[0]=
     min_min() and the remaining individuals,Gen(t)[1] to Gen(t)[k-1], are generated
     randomly;
3  Evaluate the fitness of each individual in the initial generation, i.e., compute
     Fitness(Gen(t));
4  while (the end criterion is not true) do
5  |   t ← t + 1;
6  |   Select Parent(t) from Gen(t-1);
7  |   With probability p_c, recombine individuals in Parent(t) to produce Offspr1(t);
8  |   With probability p_m, mutate individuals in Offspr1(t) to produce Offspr2(t)
   |     using VNS algorithm;
9  |   Evaluate the fitness of each individual, i.e., compute Fitness(Offspr2(t));
10 |   Replace Gen(t) from Offspr2(t) and/or Gen(t-1);
11 end
12 Find best_S which is the best solution in Gen(t);
13 return best_S;
```

Chapter 5

Experimental results

This chapter discusses the experimental results of applying the four proposed hybrid meta-heuristic algorithms, described in Chapter 4, for the static independent job scheduling in grid computing. Several experiments were carried out to analyze the performance of the proposed schedulers in terms of minimising the makespan. Three well known benchmarks with different sizes and characteristics were used in the analysis. Moreover, the performance of the proposed methods were compared against some of the best proposals described in the literature.

The development language and the other measures used to report the achieved results are described in the first section. The second section describes the essential experiments carried out to select the best parameters for each proposed method to best optimize its performance. The following sections report and discuss the experimental results achieved form applying the loosely and strongly coupled versions of the proposed methods for each of the three benchmarks. The reported results include the achieved makespan results (best, average and standard deviation), performance comparisons and the improvement of each proposed method over selected algorithms described in the literature, the statistical analysis of the performance of the proposed methods, and the average gaps of each problem instance to its lower bound. Furthermore, results summary is also provided, in which the performance of the proposed methods over the min-min method in terms of consistency is discussed.

5.1 Development tools

The Java language was used to implement the proposed methods in this research. An Intel i5-4570 CPU @ 3.20 GHz PC with 8 GB RAM has been used to carry out all the experiments reported in this thesis.

In addition to the best, average, and standard deviations, the two samples with unequal variants t-test, which was used to test the hypothesis that two samples have equal means,

was performed to statistically analyse the performance of the proposed methods with a confidence interval of 95%. Moreover, two measures will also be used to compare the results obtained by applying the proposed hybrid methods and some of the methods described in the literature. The first measure is the improvement percentage of one algorithm over another, which can be computed using Equation 5.1.

$$Improvement(\%) = \frac{Approach1 - Approach2}{Approach1} * 100\% \tag{5.1}$$

where Approach1 and Approach2 are the makespan values of the two different approaches. The second measure is the relative gap value of any approach with respect to the corresponding lower bound, which can be computed using Equation 5.2.

$$Gap = \frac{R - LB}{LB} \tag{5.2}$$

where R represents the makespan time (best or average) achieved by the proposed approach for the corresponding problem instance and the LB is the lower bound of the problem.

5.2 Parameter tuning

In order to perform parameter tuning, a fix set of parameters was selected from the literature for each of the proposed algorithms. The parameter tuning experiments were carried out using a number of instances with diverse characteristics. For the proposed VNS, the examined parameter was the order of neighbourhood structures only. Population size, α, β, pheromone evaporation rate (ρ) were among the tested parameters for the proposed hybrid ACO+VNS and ACO(VNS). On the other hand, the following parameters were examined for the proposed hybrid GA+VNS and GA(VNS): population size, crossover type, mutation type, crossover probability and mutation probability. To select the best parameter values, each algorithm was executed 30 times for each ETC instance and for each parameter, and their average was reported.

5.2.1 Parameter tuning for VNS

One of the main advantages of VNS is that it does not need many parameters. The stopping condition is the maximum number of iterations, which was set to 5. As mentioned earlier, the order of neighbourhood structures will be the main parameter that will be examined, as the forward VNS version is used in this study which means that VNS starts with k=1 and then increases k by one if no improvement is found; otherwise, set k=1. Since we have four different structures, we then have 24 possible combinations. Table 5.1 illustrates the effects of changing the order of neighbourhood structures based on different instances

with different characteristics. The tables show that case 24 was the best order recorded in almost all the tested cases.

5.2.2 Parameter tuning for ACO+VNS and ACO(VNS)

Three parameters have been examined for the hybrid ACO+VNS and ACO(VNS) which are the population size, the values of α and β and the value of ρ. The population size is set to 2 due to the attempt to reduce the computational time needed to construct solutions by ants and increase the number of generations. Various studies have suggested optimal values for α and β which vary between 1 and 10, while the suggested values for ρ were between 0.5 and 0.7 [135] [147] [60]. Therefore, three values have been used for α and β, which are 1, 5 and 10. The results indicate that α=10 and β=1 represented the best combination, as shown in Table 5.2; similarly, two values were used for ρ, which are 0.5 and 0.7. The best makespan values were achieved when using $\rho = 0.7$.

5.2.3 Parameter tuning for GA+VNS and GA(VNS)

Four parameters were tested for the hybrid GA+VNS and GA(VNS) algorithms which included the population size, crossover type, mutation type, crossover probability and mutation probability.

The candidate values for the population size were 10, 20 and 30 individuals. The best results were indicated when using a population size of 20 solutions. The experiments showed a very slow improvement rate and that a greater computational time was required to find a good mapping of jobs to resources when increasing the number of individuals from 20 to 30, suggesting that using a large population size is not beneficial for the GA+VNS and GA(VNS).

As mentioned earlier, six different crossover operators were used, which were one-point crossover (1P), two-point crossover (2P), half uniform crossover (HUX), Order Crossover (OX), Partially Matched Crossover (PMX) and Cycle Crossover (CX) with the best results being achieved when using the two-point crossover operator, as illustrated in Fig. 5.1, while Table 5.3 reports the values of other parameters used to compare the performance of different crossover operators.

Table 5.1 Neighbourhood structures order testing for GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS). The best average makespan results are reported in bold.

Case	Nieghbourhood order	u_c_hihi.0				u_i_lohi.0				u_s_lolo.0			
		GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
1	LMMTM-RMMTM-PTM-PSM	7544277.98	7540254.93	7523303.74	7418202.08	105246.31	104272.71	103461.03	102236.81	3486.33	3484.05	3481.22	3440.73
2	LMMTM-RMMTM-PSM-PTM	7490010.35	7480112.53	7461538.54	7411587.14	105079.00	104117.79	103078.38	102196.96	3479.27	3463.12	3457.79	3439.49
3	LMMTM-PTM-RMMTM-PSM	7647607.30	7637632.35	7611599.66	7429102.55	106400.19	105206.29	103997.77	102223.43	3485.33	3480.16	3478.69	3439.89
4	LMMTM-PTM-PSM-RMMTM	7550272.66	7550007.80	7524070.13	7418272.19	106126.33	105066.97	103482.83	102201.54	3487.62	3484.80	3482.79	3440.83
5	LMMTM-PSM-RMMTM-PTM	7643180.88	7619404.31	7600087.64	7427923.88	106299.66	105097.00	103967.27	102249.57	3485.46	3481.97	3478.99	3440.26
6	LMMTM-PSM-PTM-RMMTM	7480174.58	7470033.63	7459931.99	7410696.30	105088.57	104134.19	103198.71	102181.12	3479.15	3463.01	3457.50	3439.48
7	RMMTM-LMMTM-PTM-PSM	7570322.45	7566018.57	7551520.11	7419236.34	107484.99	106286.52	105017.89	102217.07	3484.10	3474.19	3469.67	3439.62
8	RMMTM-LMMTM-PSM-PTM	7500022.85	7480122.01	7462071.33	7415678.69	105079.79	104127.61	103116.24	102191.24	3479.43	3463.13	3458.66	3439.54
9	RMMTM-PTM-LMMTM-PSM	7636052.58	7612091.72	7599243.67	7426828.91	107536.67	106327.81	105076.61	102289.43	3485.17	3476.20	3474.15	3439.78
10	RMMTM-PTM-PSM-LMMTM	7630608.16	7602619.78	7594642.93	7426386.82	106349.64	105151.05	103994.63	102199.67	3486.17	3482.77	3479.86	3440.48
11	RMMTM-PSM-LMMTM-PTM	7577788.24	7570272.39	7553116.25	7419881.73	107450.44	106246.19	105017.03	102213.49	3485.08	3475.01	3471.81	3439.71
12	RMMTM-PSM-PTM-LMMTM	7481018.82	7480046.46	7460072.38	7411282.59	105102.58	104153.77	103203.13	102180.02	3478.77	3462.36	3456.87	3439.39
13	PTM-LMMTM-RMMTM-PSM	7627288.68	7601868.98	7591135.20	7426360.38	106442.55	105209.27	104000.94	102228.86	3485.78	3482.54	3479.76	3440.47
14	PTM-LMMTM-PSM-RMMTM	7614401.57	7583338.88	7561004.72	7421789.80	107296.90	106225.60	104996.28	102220.94	3485.16	3475.04	3472.56	3439.76
15	PTM-RMMTM-LMMTM-PSM	7624821.07	7600995.31	7589684.18	7426281.88	107085.27	106016.87	104002.36	102197.04	3485.63	3482.40	3479.01	3440.33
16	PTM-RMMTM-PSM-LMMTM	7583443.79	7570748.48	7556095.14	7420174.26	106332.95	105111.21	103983.93	102232.20	3486.29	3483.09	3480.05	3440.64
17	PTM-PSM-LMMTM-RMMTM	7540902.41	7530594.14	7515881.57	7416901.61	105223.71	104267.58	103435.80	102185.29	3495.14	3485.83	3483.89	3440.93
18	PTM-PSM-RMMTM-LMMTM	7539629.61	7530338.94	7513822.99	7416602.01	106114.42	105060.66	103479.53	102183.93	3488.14	3485.23	3483.81	3440.92
19	PSM-LMMTM-RMMTM-PTM	7606316.28	7577019.81	7559978.72	7421020.83	107680.08	106381.13	105089.96	102184.40	3485.02	3474.90	3470.11	3439.67
20	PSM-LMMTM-PTM-RMMTM	7585648.37	7574492.04	7558180.40	7420258.33	107176.54	106102.10	104980.12	102193.77	3483.07	3471.07	3469.33	3439.57
21	PSM-RMMTM-LMMTM-PTM	7543328.46	7540009.58	7519709.81	7417780.11	105154.46	104220.82	103431.85	102221.50	3487.60	3484.15	3482.50	3440.77
22	PSM-RMMTM-PTM-LMMTM	7541057.58	7530630.80	7516257.85	7417718.21	105151.97	104214.04	103405.53	102190.62	3487.37	3484.12	3481.77	3440.74
23	PSM-PTM-LMMTM-RMMTM	7470065.25	**7460003.97**	7458970.91	7410640.19	105025.38	104062.28	102989.42	**102178.33**	3477.79	3462.28	3455.84	3439.29
24	PSM-PTM-RMMTM-LMMTM	**7470041.77**	7460009.65	**7457700.08**	**7410271.22**	**105017.88**	**104027.98**	**102986.64**	102179.87	**3476.13**	**3462.21**	**3455.20**	**3439.13**

Table 5.2 Parameter tuning for ACO+VNS and ACO(VNS) algorithms: α and β. The best average makespan results are reported in bold.

Case	α	β	ACO+VNS			ACO(VNS)		
			u_c_hihi.0	u_i_lohi.0	u_s_lolo.0	u_c_hihi.0	u_i_lohi.0	u_s_lolo.0
1	1	1	7774955.69	107085.27	3745.30	7740919.96	105896.57	3728.38
2	1	5	7802955.98	107176.54	3755.12	7751098.51	106078.23	3733.52
3	1	10	7900218.46	107715.58	3765.21	7892111.31	106829.59	3737.49
4	5	1	7570272.39	104305.04	3598.17	7565030.42	103716.63	**3576.45**
5	5	5	7661298.85	106016.87	3634.45	7653345.35	105214.28	3625.24
6	5	10	7758704.71	106906.18	3663.78	7703753.35	105476.08	3629.75
7	10	1	**7527022.18**	**104153.77**	**3594.69**	**7519709.81**	**103570.61**	3576.83
8	10	5	7591211.43	104214.04	3601.58	7588542.95	103586.01	3583.05
9	10	10	7619404.31	105017.88	3608.22	7612251.74	104996.28	3601.12

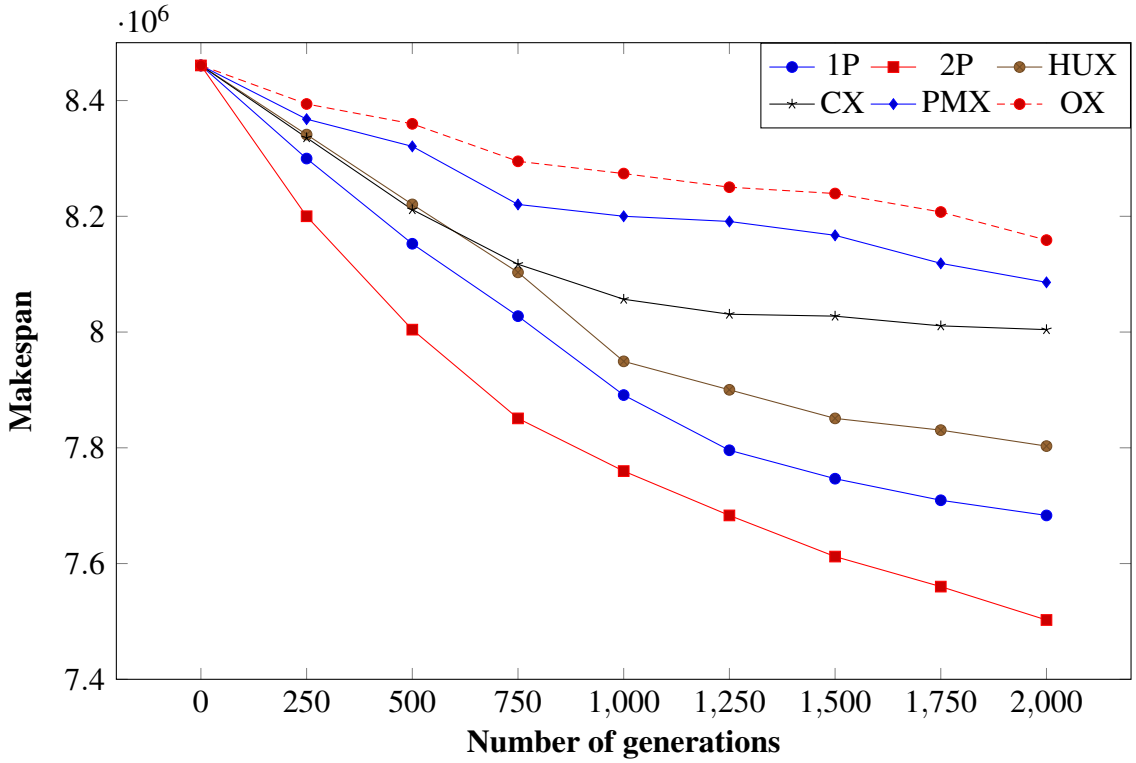

Fig. 5.1 Parameter tuning for different crossover operators of GA(VNS) using u-c-hihi.0 instance from the 512x16 dataset.

Table 5.3 Parameter values used for comparing the performance of different crossover operators.

Seeding method	min-min algorithm
Number of generations	2000
Probability of crossover	0.7
Population size	20
Selection operator	N-Tournament, N = 4
Mutation operator	VNS
Probability of mutation	0.8
Replacement operator	Steady-State

In similar fashion, four different mutation operators were used, which were Random move, Random swap, Best move and Best swap with the best results being achieved when using the Best swap operator, as illustrated in Fig. 5.2, while Table 5.4 reports the values of other parameters used to compare the performance of different mutation operators.

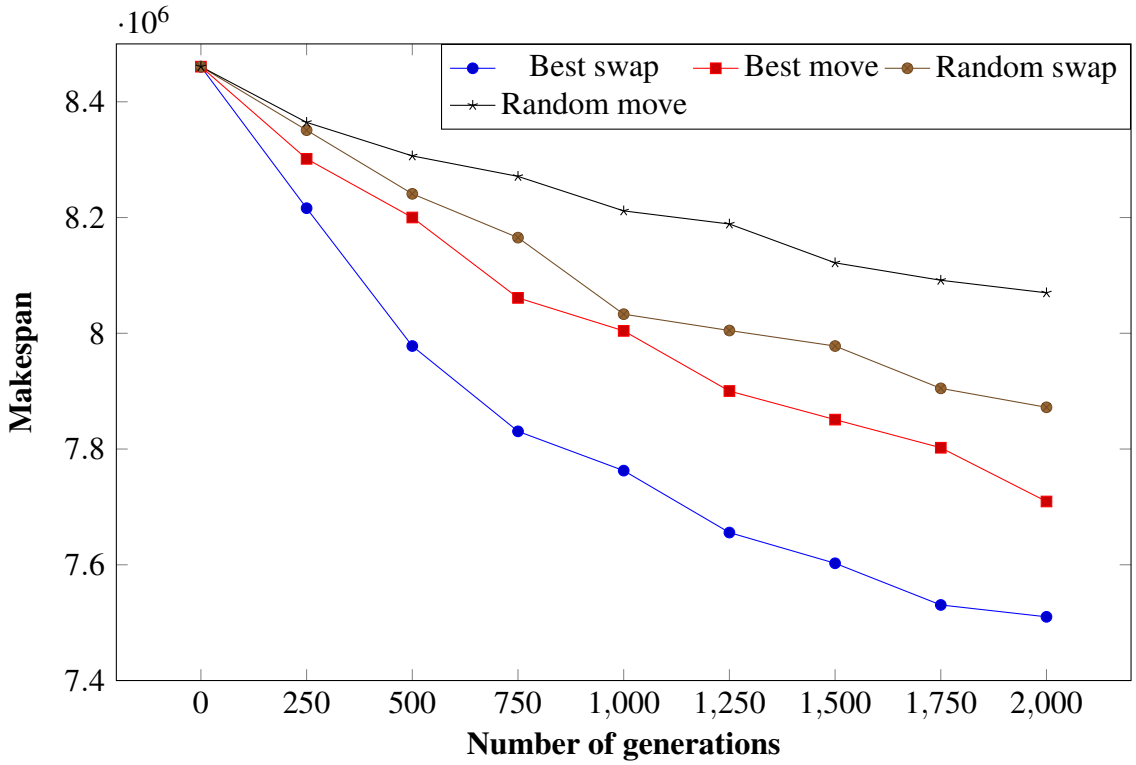

Fig. 5.2 Parameter tuning for different mutation operators of GA+VNS using u-c-hihi.0 instance from the 512x16 dataset.

Table 5.4 Parameter values used for comparing the performance of different mutation operators.

Seeding method	min-min algorithm
Number of generations	2000
Probability of crossover	0.7
Population size	20
Selection operator	N-Tournament, N = 4
Crossover operator	2P
Probability of mutation	0.8
Replacement operator	Steady-State

Finally, the probability of crossover and mutation were examined. A considerable number of studies in the literature suggested high crossover and mutation probabilities [25] [170] [169] [163]; therefore, the candidate values used were 0.7, 0.8 and 0.9. The best result was recorded when using pc = 0.7 and pm = 0.8, as shown in Fig. 5.3 and Fig. 5.4.

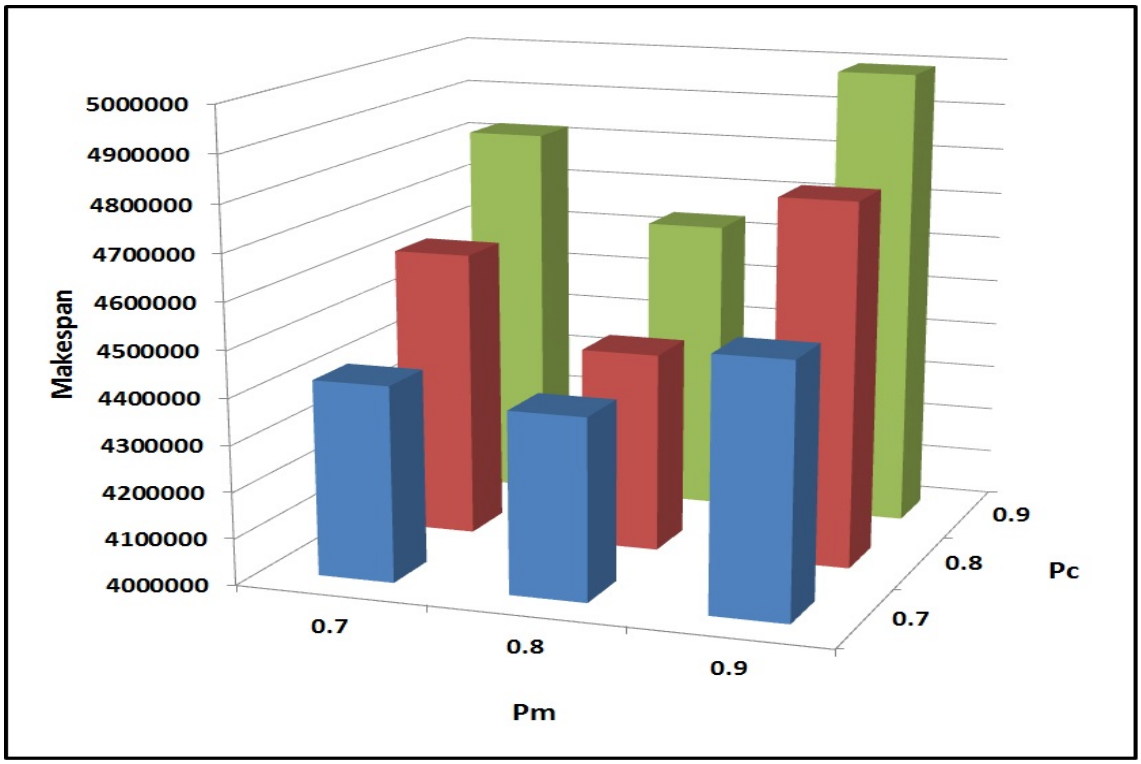

Fig. 5.3 Analysis of GA+VNS operators probabilities using u-s-hihi.0 instance from the 512x16 dataset.

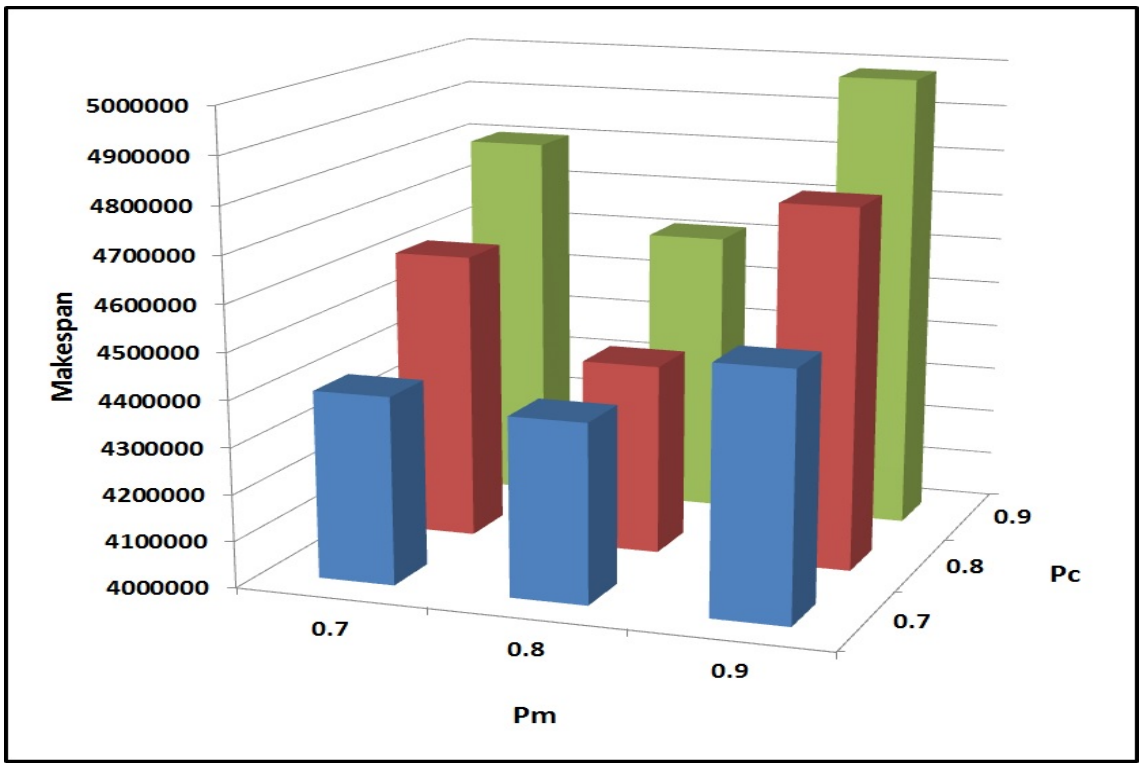

Fig. 5.4 Analysis of GA(VNS) operators probabilities using u-s-hihi.0 instance from the 512x16 dataset.

5.3 Results for instances from Liu *et al.* [93]

To enable a fair comparison, the proposed strongly coupled methods used the same number of iterations as used in [93], which is (50 x the number of jobs x the number of resources) iterations, as a stopping condition. On the other hand, the proposed loosely coupled methods used as a stopping condition (40 x the number of jobs x the number of resources) iterations for the first algorithm, i.e. GA or ACO, followed by (10 x the number of jobs x the number of resources) iterations for the VNS algorithm.

To obtain the best, average, standard deviation, and the processing time, each algorithm was executed 10 times for each instance which is also the same number used by the authors. The four proposed hybrid methods, GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS), were compared against selected algorithms from the literature. In particular, the following algorithms were selected for comparison: min-min algorithm [77], Genetic Algorithm (GA) [93], Simulated Annealing (SA) [93], Particle Swarm Optimisation (PSO) [93], Differential Evolution algorithm (DE) [139] and Two-Phase Variable Neighbourhood Search (TPVNS) [138]. All the competing algorithms were implemented sequentially. Moreover, all the above are stand-alone meta-heuristic algorithms with the exception

of TPVNS, which is a loosely coupled hybrid meta-heuristic, and min-min, which is a deterministic heuristic method.

Table 5.5 provides the performance comparison between the four proposed hybrid methods and other methods from the literature in terms of makespan. In Table 5.5, the first column represents the algorithm applied, the second column represents the criteria used in comparison, namely Res (Result for deterministic min-min algorithm), Avg (average), time (in seconds), Best (best makespan found) and σ (standard deviation). There is no information provided about the best makespan achieved by the algorithms proposed in [93] and [139], the standard deviation of the algorithm suggested in [139], or the time the algorithm proposed in [138] needed to finish. The third, fourth, fifth, and sixth columns represent the four different instances. The best results are indicated in bold.

The results in Table 5.5 show clearly that the proposed strongly coupled GA(VNS) outperforms the other approaches, including ACO(VNS), in three instances, namely (3, 13), (8, 60) and (10, 50), while PSO [26] outperforms the other methods in the (5, 100) instance. It also shows the standard deviations of the makespans of the solutions achieved by GA(VNS) are very small, which means that the algorithm can achieve a high-quality makespan in any single execution. Furthermore, it shows the time needed to finish the search process, which clearly indicates that the proposed GA(VNS) is the fastest of the meta-heuristic methods. On the other hand, the table shows the performance of the strongly couple ACO(VNS), which showed the second-best performance after GA(VNS). However, the time needed to find these results was longer than that for GA(VNS). In general, the relative performance order of the implemented methods from best to worst was: (1)GA(VNS), (2)ACO(VNS), (3)ACO+VNS, and (4)GA+VNS.

Tables 5.6, 5.7, 5.8 and 5.9 show the improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS), respectively, over the state-of-the-art methods described in the literature. Table 5.10 presents the improvement percentages of GA(VNS) over GA+VNS, ACO+VNS and ACO(VNS), which clearly indicates that GA(VNS) performs better than the rest. Furthermore, the two sample t-test with unequal variants was performed to statistically analyse the performance of the four proposed hybrid methods. Table 5.10 reports the corresponding p-value for each problem instance, all of which were less than 0.05; hence, we can reject the null hypothesis and consider the improvements in the makespan to be statistically significant. Fig. 5.5 shows the overall improvements of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) over the deterministic heuristic min-min algorithm for the problem instances of Liu *et al.*

Table 5.5 Makespan results for dataset instances from Liu *et al.* [93].

Algorithm		(3, 13)	(5, 100)	(8, 60)	(10, 50)
min-min[77]	Res	56.0000	87.6693	47.8764	42.7346
	time	0.0001	0.0010	0.0010	0.0020
GA[93]	Avg	47.1167	85.7431	42.9270	38.0428
	σ	0.7700	0.6217	0.4150	0.6613
	time	302.9210	2415.9000	2263.0000	2628.1000
SA[93]	Avg	46.6000	90.7338	55.4594	41.7889
	σ	0.4856	6.3833	2.0605	8.0773
	time	332.5000	6567.8000	6094.9000	6926.4000
PSO[93]	Avg	46.2667	**84.0544**	41.9489	37.6668
	σ	0.2854	0.5030	0.6944	0.6068
	time	106.2030	1485.6000	1521.0000	1585.7000
DE[139]	Avg	46.0500	86.3600	42.4800	38.3900
	time	22.4400	1550.3227	430.0000	285.2600
TPVNS[138]	Best	**46.0000**	85.4345	41.7227	35.1586
	Avg	46.2500	85.4357	41.7412	35.2478
	σ	0.1100	0.1000	0.1200	0.1300
GA+VNS	Best	**46.0000**	85.5295	41.6551	35.1525
	Avg	**46.0000**	85.5401	41.7027	35.2268
	std dev	0.0000	0.0152	0.0459	0.0643
	time	4.3614	194.2657	76.1913	70.2563
ACO+VNS	Best	**46.0000**	85.5284	41.5905	35.1495
	Avg	**46.0000**	85.5352	41.6311	35.1996
	σ	0.0000	0.0072	0.0412	0.0564
	time	7.1557	742.8736	387.9572	379.6415
ACO(VNS)	Best	**46.0000**	85.5281	41.5853	35.1447
	Avg	**46.0000**	85.5283	41.5918	35.1618
	σ	0.0000	0.0002	0.0077	0.0215
	time	7.2638	751.2500	392.7000	381.2500
GA(VNS)	Best	**46.0000**	85.5279	**41.5795**	**35.1365**
	Avg	**46.0000**	85.5281	**41.5803**	**35.1382**
	σ	0.0000	0.0002	0.0011	0.0022
	time	4.3340	140.6454	76.0624	70.3302

Table 5.6 Average improvement percentages of GA+VNS over selected methods from the literature for dataset instances from Liu *et al.* [93].

Instance	Min-min	GA	SA	PSO	DE	TPVNS
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405
(5, 100)	2.4287	0.2368	5.7241	-1.7675	0.9494	-0.1222
(8, 60)	12.8951	2.8521	24.8050	0.5869	1.8298	0.0922
(10, 50)	17.5684	7.4022	15.7030	6.4779	8.2396	0.0596
Avg	12.6873	3.2153	11.8799	1.4684	2.7819	0.1425

Table 5.7 Average improvement percentages of ACO+VNS over selected methods from the literature for dataset instances from Liu *et al.* [93].

Instance	Min-min	GA	SA	PSO	DE	TPVNS	GA+VNS
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405	0.0000
(5, 100)	2.4343	0.2425	5.7295	-1.7617	0.9551	-0.1165	0.0057
(8, 60)	13.0446	3.0188	24.9341	0.7576	1.9984	0.2638	0.1717
(10, 50)	17.6321	7.4737	15.7681	6.5501	8.3105	0.1367	0.0772
Avg	12.7420	3.2763	11.9298	1.5306	2.8431	0.2061	0.0637

Table 5.8 Average improvement percentages of ACO(VNS) over selected methods from the literature for dataset instances from Liu *et al.* [93].

Instance	Min-min	GA	SA	PSO	DE	TPVNS	GA+VNS	ACO+VNS
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405	0.0000	0.0000
(5, 100)	2.4421	0.2505	5.7371	-1.7535	0.9631	-0.1084	0.0138	0.0081
(8, 60)	13.1268	3.1105	25.0050	0.8514	2.0910	0.3580	0.2660	0.0945
(10, 50)	17.7204	7.5729	15.8584	6.6503	8.4089	0.2439	0.1844	0.1073
Avg	12.7866	3.3260	11.9720	1.5812	2.8929	0.2585	0.1161	0.0525

Table 5.9 Average improvement percentages of GA(VNS) over selected methods from the literature for dataset instances from Liu *et al.* [93].

Instance	Min-min	GA	SA	PSO	DE	TPVNS	GA+VNS	ACO+VNS	ACO(VNS)
(3, 13)	17.8571	2.3701	1.2876	0.5764	0.1086	0.5405	0.0000	0.0000	0.0000
(5, 100)	2.4424	0.2508	5.7374	-1.7532	0.9633	-0.1081	0.0141	0.0083	0.0003
(8, 60)	13.1507	3.1371	25.0257	0.8786	2.1179	0.3854	0.2935	0.1220	0.0275
(10, 50)	17.7759	7.6352	15.9151	6.7132	8.4706	0.3111	0.2517	0.1746	0.0674
Avg	12.8065	3.3483	11.9914	1.6038	2.9151	0.2822	0.1398	0.0762	0.0238

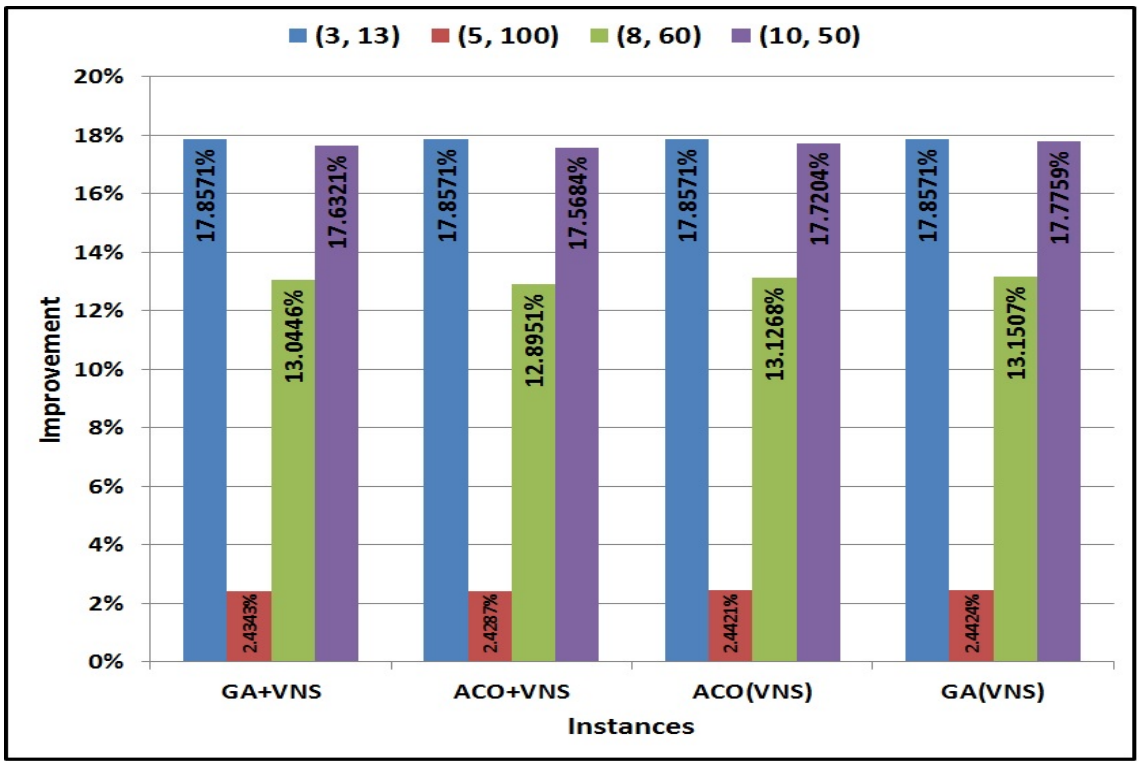

Fig. 5.5 GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) improvement percentages with respect to the min-min heuristic for Liu *et al.* dataset.

Table 5.10 Average improvement percentages and statistical analysis of GA(VNS) over other proposed methods for dataset instances from Liu *et al.* [93].

Instance	GA+VNS		ACO+VNS		ACO(VNS)	
	improvement	p-value	improvement	p-value	improvement	p-value
(3, 13)	0.0000	not valid	0.0000	not valid	0.0000	not valid
(5, 100)	0.0141	$<10^{-5}$	0.0083	$<10^{-5}$	0.0003	0.012170
(8, 60)	0.2935	$<10^{-5}$	0.1220	$<10^{-5}$	0.0275	$<10^{-5}$
(10, 50)	0.2517	$<10^{-5}$	0.1746	$<10^{-5}$	0.0674	$<10^{-4}$
Avg	0.1398		0.0762		0.0238	

5.4 Results for instances from Braun *et al.* [19]

The second dataset involves the classical 12 problem instances of Braun *et al.* Each instance has 512 jobs and 16 resources. The four proposed hybrid methods, GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS), were compared against selected algorithms from the literature, namely min-min algorithm [77], Genetic Algorithm (GA) [19], Cellular Memetic Algorithms (cMA) [165], Memetic Algorithm and Tabu Search (MA+TS) [160],

Ant Colony Optimization and Tabu Search (ACO+TS) [135], Tabu Search (TS) [171], parallel Cross generational Heterogeneous recombination Cataclysmic mutation (pCHC) [117], and Two-Phase Variable Neighbourhood Search (TPVNS) [138]. All the competing algorithms were implemented sequentially with the exception of pCHC which was implemented using the parallel mode. Moreover, all these are loosely coupled hybrid meta-heuristics apart from min-min, which is a deterministic heuristic method. Each algorithm uses different stopping times and different number of executions. For each problem instance, the deterministic min-min algorithm requires less than one second to finish the mapping, while GA, ACO-VNS, TS needed 65 s, 3.5 h, and 100 s, respectively, and the average results were reported after 100, 1, and 10 runs, respectively. cMA, MA+TS, pCHC and TPVNS required 90 s to find the solution and the average results were achieved after 10, 10, 50 and 50 runs, respectively. In this thesis, the GA-based methods were allowed to run for 90 seconds while the ACO-based methods needed 9 minutes as shown in Table 5.11. To obtain the best, average and standard deviation values, GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) were executed 50 times for each problem instance.

Table 5.11 Stopping times for the proposed methods for 512x16 dataset instances from Braun *et al.*

Algorithm	stopping time	total time
GA(VNS)	90 seconds	90 seconds
GA+VNS	80 seconds for GA 10 seconds for VNS	90 seconds
ACO(VNS)	9 minutes	9 minutes
ACO+VNS	8 minutes and 50 seconds for ACO 10 seconds for VNS	9 minutes

Table 5.12 provides the results of applying GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) compared to the selected methods. The best results are indicated in bold, which show clearly that the proposed strongly coupled GA(VNS) outperforms all other approaches in all instances. The GA(VNS) algorithm is expected to find high-quality schedules in any single execution since it has very small standard deviation values in the range [0.01, 0.07]. The table also shows the results for the proposed strongly coupled ACO(VNS) which showed the second-best performance after GA(VNS) with relatively small standard deviation values between 0.03 and 1.0. Although it needed six times longer than GA(VNS), it is typically 140 times faster than ACO+TS [135], outperforming it in 10 out of 12 instances. In general, the relative performance order of the implemented methods from best to worst was: (1)GA(VNS), (2)ACO(VNS), (3)ACO+VNS, and (4)GA+VNS.

Tables 5.13, 5.14, 5.15 and 5.16 show the improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) respectively over the selected methods from the

Table 5.12 Makespan results for 512x16 dataset instances from Braun *et al.*

Instance	min-min[77]	GA[19]	cMA[165]	MA+TS[160]	ACO+TS[135]	TS[171]	pCHC[117]		TPVNS[138]		GA+VNS			ACO+VNS			ACO(VNS)			GA(VNS)			LB
	res	avg	avg	avg	best	avg	best	avg	best	avg	best	avg	σ	best	avg	σ	best	avg	σ	best	avg	σ	
u_c_hihi.0	8460675.00	8050844.50	7700929.80	7530020.20	7497200.90	7448640.50	7461819.10	7481194.50	7439471.80	7494257.80	7440003.55	7467089.73	0.26%	7431111.34	7453099.03	0.21%	7429322.59	7452836.64	0.20%	**7391226.27**	**7403808.91**	0.07%	7346524.20
u_c_hilo.0	164022.44	156249.20	155334.80	153917.20	154234.60	153263.30	153791.90	153924.00	153270.10	154400.30	154023.37	154486.23	0.23%	153249.98	153713.01	0.23%	153183.78	153597.87	0.11%	**153106.67**	**153123.67**	0.01%	152700.40
u_c_lohi.0	275837.34	258756.80	251360.20	245288.90	244097.30	241672.70	241524.00	243446.30	240803.30	244043.20	241008.49	243831.10	0.25%	240500.86	242991.77	0.22%	239259.98	240342.04	0.20%	**239258.20**	**239289.10**	0.01%	238138.10
u_c_lolo.0	5546.26	5272.30	5218.20	5173.70	5178.40	5155.00	5177.50	5181.60	5154.80	5190.30	5170.04	5190.02	0.22%	5153.00	5173.01	0.22%	5148.15	5163.60	0.11%	**5148.02**	**5148.59**	0.01%	5132.80
u_i_hihi.0	3513919.25	3104762.50	3186664.70	3058474.90	2947754.10	2957854.10	2952493.20	2956905.70	2944074.60	2955764.70	2950248.73	2959773.53	0.18%	2942987.16	2955283.00	0.23%	2938416.02	2939907.97	0.03%	**2938380.62**	**2939301.25**	0.02%	2909326.60
u_i_hilo.0	80755.68	75816.10	75856.60	75108.50	73776.20	73692.90	73639.80	73847.10	73378.00	73927.00	73600.17	73837.88	0.17%	73374.91	73625.74	0.18%	73377.99	73512.45	0.05%	**73362.98**	**73391.97**	0.02%	73057.90
u_i_lohi.0	120517.71	107500.70	110620.80	105808.60	102445.80	103865.70	102136.10	102677.30	102057.50	102599.70	102111.71	102593.48	0.21%	102055.23	102554.42	0.22%	102052.98	102452.20	0.18%	**102051.74**	**102167.07**	0.05%	101063.40
u_i_lolo.0	2779.09	2614.40	2624.20	2596.60	2553.50	2552.10	2549.80	2557.20	2547.90	2560.10	2546.98	2556.89	0.18%	2543.60	2552.02	0.16%	2541.52	2548.78	0.13%	**2541.17**	**2542.98**	0.03%	2529.00
u_s_hihi.0	5160343.00	4566206.00	4424540.90	4321015.40	4162547.90	4168795.90	4198779.50	4239146.30	4145941.70	4197996.50	4146224.92	4195105.32	0.26%	4144999.88	4167862.59	0.30%	4106391.16	4140091.67	0.35%	**4102908.20**	**4104613.59**	0.02%	4063563.70
u_s_hilo.0	104540.73	98519.40	98283.70	97177.30	96762.00	96180.90	96623.30	96750.30	95872.30	96330.40	96402.20	96608.24	0.20%	96118.69	96178.73	0.06%	96117.37	96167.89	0.03%	**95788.97**	**95823.04**	0.02%	95419.00
u_s_lohi.0	140284.48	130616.50	130014.50	127633.00	123922.00	123407.40	123251.50	123989.40	122986.00	123954.30	124006.42	125869.01	0.88%	123595.16	125729.82	1.01%	123576.46	125650.31	1.00%	**122084.10**	**122143.20**	0.02%	120452.30
u_s_lolo.0	3867.49	3583.40	3522.10	3484.10	3455.20	3450.50	3450.10	3472.20	3440.50	3461.90	3460.78	3470.30	0.20%	3440.02	3450.20	0.22%	3439.35	3449.46	0.13%	**3436.63**	**3437.97**	0.01%	3414.80

bibliography. GA(VNS) shows a better improvement percentage over all the methods compared with a minimum average improvement of 0.86, while the minimum average improvement percentage of ACO(VNS), which was the second best method, was 0.30. These results indicate that the two proposed strongly coupled algorithms, ACO(VNS) and GA(VNS), represent the new state-of-the-art sequential hybrid algorithms for the job scheduling problem in grid computing. Moreover, as shown in Table 5.14, ACO+VNS also outperformed almost all other methods with the exception of TS, in which it outperformed ACO+VNS in the average improvement percentage. However, ACO+VNS outperformed TS in 7 out of 12 instances. On the other hand, GA+VNS outperformed half of the selected methods as shown in Table 5.13.

Table 5.13 Average improvement percentages of GA+VNS over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	11.74	7.25	3.04	0.84	0.40	-0.25	0.19	0.36
u_c_hilo.0	5.81	1.13	0.55	-0.37	-0.16	-0.80	-0.37	-0.06
u_c_lohi.0	11.60	5.77	3.00	0.59	0.11	-0.89	-0.16	0.09
u_c_lolo.0	6.42	1.56	0.54	-0.32	-0.22	-0.68	-0.16	0.01
u_i_hihi.0	15.77	4.67	7.12	3.23	-0.41	-0.06	-0.10	-0.14
u_i_hilo.0	8.57	2.61	2.66	1.69	-0.08	-0.20	0.01	0.12
u_i_lohi.0	14.87	4.56	7.26	3.04	-0.14	1.22	0.08	0.01
u_i_lolo.0	8.00	2.20	2.56	1.53	-0.13	-0.19	0.01	0.13
u_s_hihi.0	18.70	8.13	5.19	2.91	-0.78	-0.63	1.04	0.07
u_s_hilo.0	7.59	1.94	1.70	0.59	0.16	-0.44	0.15	-0.29
u_s_lohi.0	10.28	3.63	3.19	1.38	-1.57	-1.99	-1.52	-1.54
u_s_lolo.0	10.27	3.16	1.47	0.40	-0.44	-0.57	0.05	-0.24
Avg	10.80	3.88	3.19	1.29	-0.27	-0.46	-0.06	-0.12

The two sample t-test with unequal variants was performed to statistically analyse the performance of the four proposed hybrid methods. Table 5.17 reports the improvement percentages for GA(VNS) over GA+VNS, ACO+VNS, and ACO(VNS), which clearly indicate that GA(VNS) performs better than the rest. Moreover, Table 5.17 reports the corresponding p-value for each instance which was less than 0.05, and hence we can consider the improvement of GA(VNS) over the rest in terms of makespan to be statistically significant.

Table 5.14 Average improvement percentages of ACO+VNS over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	11.91	7.42	3.22	1.02	0.59	-0.06	0.38	0.55
u_c_hilo.0	6.29	1.62	1.04	0.13	0.34	-0.29	0.14	0.45
u_c_lohi.0	11.91	6.09	3.33	0.94	0.45	-0.55	0.19	0.43
u_c_lolo.0	6.73	1.88	0.87	0.01	0.10	-0.35	0.17	0.33
u_i_hihi.0	15.90	4.81	7.26	3.37	-0.26	0.09	0.05	0.02
u_i_hilo.0	8.83	2.89	2.94	1.97	0.20	0.09	0.30	0.41
u_i_lohi.0	14.91	4.60	7.29	3.08	-0.11	1.26	0.12	0.04
u_i_lolo.0	8.17	2.39	2.75	1.72	0.06	0.00	0.20	0.32
u_s_hihi.0	19.23	8.72	5.80	3.54	-0.13	0.02	1.68	0.72
u_s_hilo.0	8.00	2.38	2.14	1.03	0.60	0.00	0.59	0.16
u_s_lohi.0	10.38	3.74	3.30	1.49	-1.46	-1.88	-1.40	-1.43
u_s_lolo.0	10.79	3.72	2.04	0.97	0.14	0.01	0.63	0.34
Avg	11.09	4.19	3.50	1.61	0.05	-0.14	0.25	0.19

Table 5.15 Average improvement percentages of ACO(VNS) over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	11.91	7.43	3.22	1.03	0.59	-0.06	0.38	0.55
u_c_hilo.0	6.36	1.70	1.12	0.21	0.41	-0.22	0.21	0.52
u_c_lohi.0	12.87	7.12	4.38	2.02	1.54	0.55	1.28	1.52
u_c_lolo.0	6.90	2.06	1.05	0.20	0.29	-0.17	0.35	0.51
u_i_hihi.0	16.34	5.31	7.74	3.88	0.27	0.61	0.57	0.54
u_i_hilo.0	8.97	3.04	3.09	2.12	0.36	0.24	0.45	0.56
u_i_lohi.0	14.99	4.70	7.38	3.17	-0.01	1.36	0.22	0.14
u_i_lolo.0	8.29	2.51	2.87	1.84	0.18	0.13	0.33	0.44
u_s_hihi.0	19.77	9.33	6.43	4.19	0.54	0.69	2.34	1.38
u_s_hilo.0	8.01	2.39	2.15	1.04	0.61	0.01	0.60	0.17
u_s_lohi.0	10.43	3.80	3.36	1.55	-1.39	-1.82	-1.34	-1.37
u_s_lolo.0	10.81	3.74	2.06	0.99	0.17	0.03	0.65	0.36
Avg	11.30	4.43	3.74	1.85	0.30	0.11	0.50	0.44

Table 5.16 Average improvement percentages of GA(VNS) over some methods from the literature for the 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS
u_c_hihi.0	12.49	8.04	3.86	1.68	1.25	0.60	1.03	1.21
u_c_hilo.0	6.64	2.00	1.42	0.52	0.72	0.09	0.52	0.83
u_c_lohi.0	13.25	7.52	4.80	2.45	1.97	0.99	1.71	1.95
u_c_lolo.0	7.17	2.35	1.33	0.49	0.58	0.12	0.64	0.80
u_i_hihi.0	16.35	5.33	7.76	3.90	0.29	0.63	0.60	0.56
u_i_hilo.0	9.12	3.20	3.25	2.29	0.52	0.41	0.62	0.72
u_i_lohi.0	15.23	4.96	7.64	3.44	0.27	1.64	0.50	0.42
u_i_lolo.0	8.50	2.73	3.09	2.06	0.41	0.36	0.56	0.67
u_s_hihi.0	20.46	10.11	7.23	5.01	1.39	1.54	3.17	2.22
u_s_hilo.0	8.34	2.74	2.50	1.39	0.97	0.37	0.96	0.53
u_s_lohi.0	12.93	6.49	6.05	4.30	1.44	1.02	1.49	1.46
u_s_lolo.0	11.11	4.06	2.39	1.32	0.50	0.36	0.99	0.69
Avg	11.80	4.96	4.28	2.40	0.86	0.68	1.06	1.01

Table 5.17 Average improvement percentages and statistical analysis of GA(VNS) over other methods for Braun 512x16 dataset.

Instance	GA+VNS		ACO+VNS		ACO(VNS)	
	improvement	p-value	improvement	p-value	improvement	p-value
u_c_hihi.0	0.85	$<10^{-5}$	0.66	$<10^{-5}$	0.66	$<10^{-5}$
u_c_hilo.0	0.88	$<10^{-5}$	0.38	$<10^{-5}$	0.31	$<10^{-5}$
u_c_lohi.0	1.86	$<10^{-5}$	1.52	$<10^{-5}$	0.44	$<10^{-5}$
u_c_lolo.0	0.80	$<10^{-5}$	0.47	$<10^{-5}$	0.29	$<10^{-5}$
u_i_hihi.0	0.69	$<10^{-5}$	0.54	$<10^{-5}$	0.02	0.00056
u_i_hilo.0	0.60	$<10^{-5}$	0.32	$<10^{-5}$	0.16	$<10^{-5}$
u_i_lohi.0	0.42	$<10^{-5}$	0.38	$<10^{-5}$	0.28	$<10^{-5}$
u_i_lolo.0	0.54	$<10^{-5}$	0.35	$<10^{-5}$	0.23	$<10^{-5}$
u_s_hihi.0	2.16	$<10^{-5}$	1.52	$<10^{-5}$	0.86	$<10^{-5}$
u_s_hilo.0	0.81	$<10^{-5}$	0.37	$<10^{-5}$	0.36	$<10^{-5}$
u_s_lohi.0	2.96	$<10^{-5}$	2.85	$<10^{-5}$	2.79	$<10^{-5}$
u_s_lolo.0	0.93	$<10^{-5}$	0.35	$<10^{-5}$	0.33	$<10^{-5}$
Avg	1.13		0.81		0.56	

In Table 5.12, the last column represents the Lower Bound (LB) values of each problem instance, as reported in [28]. Table 5.18 summarizes the gaps between the average makespan results for the proposed methods and selected algorithms from the literature

and their corresponding lower bounds. GA(VNS) and ACO(VNS) achieved the smallest average gap values to the lower bound with 0.71 and 1.28, respectively, which indicates that the quality of the solutions they found are very high compared to the others. The average percentage gap of GA(VNS) for 512x16 problem instances was 0.71%, with 8 out of 12 instances being below 1%, while ACO(VNS) achieved an average gap percentage of 1.28% with 6 out of 12 being less than 1%. On the other hand, the average percentage gap of ACO VNS was 1.54%, with 5 out of 12 instances being below 1%, while GA+VNS achieved an average gap percentage of 1.86% with 12 out of 12 being greater than 1%.

Table 5.18 The gap values of the average makespan for the proposed methods and selected algorithms from the literature and the corresponding lower bounds for 512x16 dataset.

Instance	min-min	GA	cMA	MA+TS	ACO+TS	TS	pCHC	TPVNS	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
u_c_hihi.0	15.17	9.59	4.82	2.50	2.05	1.39	1.83	2.01	1.64	1.45	1.45	0.78
u_c_hilo.0	7.41	2.32	1.73	0.80	1.00	0.37	0.80	1.11	1.17	0.66	0.59	0.28
u_c_lohi.0	15.83	8.66	5.55	3.00	2.50	1.48	2.23	2.48	2.39	2.04	0.93	0.48
u_c_lolo.0	8.06	2.72	1.66	0.80	0.89	0.43	0.95	1.12	1.11	0.78	0.60	0.31
u_i_hihi.0	20.78	6.72	9.53	5.13	1.32	1.67	1.64	1.60	1.73	1.58	1.05	1.03
u_i_hilo.0	10.54	3.78	3.83	2.81	0.98	0.87	1.08	1.19	1.07	0.78	0.62	0.46
u_i_lohi.0	19.25	6.37	9.46	4.70	1.37	2.77	1.60	1.52	1.51	1.48	1.37	1.09
u_i_lolo.0	9.89	3.38	3.76	2.67	0.97	0.91	1.12	1.23	1.10	0.91	0.78	0.55
u_s_hihi.0	26.99	12.37	8.88	6.34	2.44	2.59	4.32	3.31	3.24	2.57	1.88	1.01
u_s_hilo.0	9.56	3.25	3.00	1.84	1.41	0.80	1.40	0.96	1.25	0.80	0.78	0.42
u_s_lohi.0	16.46	8.44	7.94	5.96	2.88	2.45	2.94	2.91	4.50	4.38	4.32	1.40
u_s_lolo.0	13.26	4.94	3.14	2.03	1.18	1.05	1.68	1.38	1.63	1.04	1.01	0.68
Avg	14.43	6.04	5.28	3.21	1.58	1.40	1.80	1.73	1.86	1.54	1.28	0.71

5.5 Results for instances from Nesmachnow *et al.* [118]

The third dataset consists of two sizes: 1024x32 and 2048x64, each of which contains 24 problem instances. Unlike the dataset of Braun *et al.* [19] , which is considered the *de facto* standard benchmark for studying the job scheduling problem in grid computing, the literature does not include much work which addresses this dataset. The performance of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) was compared against the following algorithms from the bibliography: min-min algorithm [77], parallel Cross generational Heterogeneous recombination Cataclysmic mutation(pCHC) [117], and Two-Phase Variable Neighbourhood Search (TPVNS) [138]. All the competing algorithms were implemented sequentially apart from pCHC, which was implemented using the parallel mode. Moreover, all of these are loosely coupled hybrid meta-heuristics apart from min-min, which is a deterministic heuristic method. For both sizes, the GA-based methods were allowed to run for 90 seconds, which was the same time used for pCHC and TPVNS, while the ACO-based methods ran for 9 minutes, i.e., the same stopping times parameters, that are illustrated in Table 5.11, were used. To obtain the best, average and standard deviation values, both schedulers were executed 50 times for each problem instance.

Tables 5.19 and 5.20 provide the results of applying GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) for the 1024x32 and 2048x64 dataset instances, respectively, compared to the selected methods. The best results are indicated in bold, which show clearly that GA(VNS) outperforms all other approaches in all instances. The GA(VNS) algorithm is expected to find high quality schedules in any single execution since it has very small standard deviation values, which vary between 0.02 and 0.33 for the 1024x32 dataset, and between 0.01 and 0.26 for the 2048x64 dataset. In general, for both sizes, the relative performance order of the implemented methods from best to worst was: (1)GA(VNS), (2)ACO(VNS), (3)ACO+VNS, and (4)GA+VNS.

Tables 5.21 and 5.22 show the percentages of makespan reduction for GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) for the 1024x32 and 2048x64 dataset instances, respectively, over the selected methods from the bibliography. For both sizes, GA(VNS) shows a better improvement percentage over all the compared methods. ACO(VNS) achieved the second-best improvement percentages for the 1024x32 dataset; however, it was only the third-best results for the 2048x64 dataset as TPVNS achieved a slightly better average improvement percentage. ACO+VNS and GA+VNS achieved the third and fourth -best improvement percentages for the 1024x32 dataset, respectively; however, they were only the fourth and fifth -best results for the 2048x64 dataset as TPVNS achieved a slightly better average improvement percentage.

Tables 5.23 and 5.24 present the improvement percentages for GA(VNS) over GA+VNS, ACO+VNS and ACO(VNS) for the 1024x32 and 2048x64 dataset instances, respectively, which clearly indicate that GA(VNS) performs better than the rest. Moreover, the table reports the corresponding p-value when applying the two sample t-test with unequal variants for each instance to statistically analyse the performance of the four proposed hybrid methods. The p-values were less than 0.05, and hence, we can consider the improvement of GA(VNS) over the other methods in terms of makespan to be statistically significant.

Table 5.19 Makespan results for 1024x32 dataset instances.

Instance	min-min[77]	pCHC[117]		TPVNS[138]		GA+VNS			ACO+VNS			ACO(VNS)			GA(VNS)			LB
		Best	Avg	Best	Avg	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	
A.u_c_hihi	22508062.40	20327924.00	20510300.90	20194902.00	20284191.00	20000839.81	20120970.88	0.36%	19900406.82	19914604.19	0.04%	19800979.10	19818850.81	0.04%	**19602098.31**	**19636595.22**	0.07%	19449230.00
A.u_c_hilo	2255966.00	2048582.70	2058352.20	2046648.00	2050942.20	2032610.46	2057658.85	0.99%	2020675.10	2049010.43	1.12%	2004755.76	2033689.80	1.01%	**1957526.50**	**1959206.37**	0.02%	1951345.00
A.u_c_lohi	2155.00	1956.70	2000.00	1962.00	1970.20	1961.51	1969.24	0.30%	1955.85	1964.30	0.33%	1900.08	1905.64	0.15%	**1876.53**	**1880.02**	0.06%	1866.40
A.u_c_lolo	225.90	207.50	217.80	206.70	213.40	207.04	210.73	0.10%	204.08	208.18	0.12%	200.00	200.25	0.06%	**199.72**	**200.02**	0.08%	198.90
A.u_i_hihi	6367767.60	5169960.50	5244046.90	5167781.00	5221702.00	5170416.45	5198152.08	0.22%	5150630.96	5156940.77	0.03%	5100026.33	5104782.53	0.04%	**5052814.63**	**5085838.69**	0.20%	5012207.00
A.u_i_hilo	641438.40	490280.30	492699.40	489525.20	493800.10	490062.73	492357.85	0.21%	487035.99	491032.32	0.19%	485003.80	485316.32	0.03%	**478681.94**	**480597.47**	0.18%	474404.60
A.u_i_lohi	664.70	518.20	523.60	522.40	530.10	525.04	529.12	0.22%	521.95	527.05	0.27%	520.02	520.96	0.09%	**507.63**	**509.85**	0.19%	503.40
A.u_i_lolo	63.70	50.60	51.70	50.80	51.90	50.90	51.95	0.30%	50.71	51.61	0.25%	50.00	50.27	0.26%	**49.43**	**49.61**	0.19%	49.00
A.u_s_hihi	14125881.60	12243560.00	12439843.10	12155750.00	12306122.00	12191093.63	12288240.81	0.18%	12141615.99	12245458.13	0.08%	12000006.48	12011669.03	0.06%	**11690402.28**	**11730490.65**	0.11%	11553632.00
A.u_s_hilo	1319050.60	1187506.40	1214303.00	1175338.00	1185443.20	1174150.66	1184814.83	0.17%	1173102.71	1182011.91	0.14%	1170121.89	1175301.84	0.34%	**1137089.07**	**1143819.59**	0.17%	1126556.00
A.u_s_lohi	1380.50	1186.80	1199.20	1184.80	1194.60	1185.14	1193.33	0.21%	1181.58	1187.11	0.14%	1170.02	1173.05	0.13%	**1131.06**	**1135.86**	0.20%	1122.20
A.u_s_lolo	140.60	122.40	126.50	122.00	123.10	123.01	125.45	0.42%	121.22	123.06	0.32%	120.03	122.14	0.98%	**117.49**	**117.72**	0.17%	116.70
B.u_c_hihi	6708228.50	6169823.00	6200118.00	6189681.00	6200401.50	6188015.44	6250005.32	1.16%	6160849.71	6195851.28	0.65%	6081103.22	6104938.95	0.28%	**6003087.48**	**6008676.39**	0.05%	5980872.00
B.u_c_hilo	66684.50	61114.70	61390.10	60807.50	61599.20	61001.55	61523.19	0.19%	60765.99	61466.78	0.26%	60006.62	60341.92	0.42%	**59287.94**	**59378.59**	0.06%	58942.50
B.u_c_lohi	232011.80	215149.20	218124.80	214387.10	216481.50	214900.17	216319.18	0.21%	214059.11	215024.18	0.14%	210001.28	210060.30	0.02%	**209183.02**	**209701.15**	0.07%	207892.80
B.u_c_lolo	2386.30	2164.30	2208.40	2142.10	2158.30	2146.98	2157.95	0.13%	2131.66	2152.41	0.24%	2100.04	2103.13	0.08%	**2084.10**	**2087.69**	0.07%	2078.00
B.u_i_hihi	2164576.70	1630288.60	1670112.70	1626086.00	1628729.60	1627286.48	1630179.33	0.32%	1625061.53	1627975.62	0.32%	1620135.38	1625042.35	0.14%	**1586928.39**	**1601151.49**	0.31%	1567179.00
B.u_i_hilo	17083.10	15121.50	15464.10	15003.10	15715.80	15120.30	15452.32	0.34%	15003.02	15303.88	0.09%	15001.07	15059.49	0.16%	**14702.66**	**14848.90**	0.33%	14582.30
B.u_i_lohi	56601.20	49569.90	50128.20	49264.10	49981.20	49850.73	50263.48	0.24%	49129.13	49903.48	0.26%	48912.32	49431.40	0.81%	**48101.79**	**48259.18**	0.18%	47606.90
B.u_i_lolo	585.00	496.10	507.40	492.70	501.40	494.23	502.05	0.26%	492.02	500.09	0.17%	490.18	491.68	0.13%	**482.27**	**485.53**	0.27%	477.40
B.u_s_hihi	3967265.90	3393010.20	3430218.10	3344875.00	3392157.30	3342051.23	3389658.57	0.34%	3334000.81	3370831.23	0.13%	3300418.86	3328580.56	0.54%	**3217747.08**	**3248005.73**	0.33%	3178482.00
B.u_s_hilo	40691.60	35988.40	36515.60	35352.20	36911.20	35344.07	36507.23	0.36%	35200.70	35916.75	0.10%	35000.62	35040.84	0.06%	**34285.87**	**34705.80**	0.32%	33948.70
B.u_s_lohi	135624.60	115179.20	118070.30	114653.30	117017.10	114726.42	117521.08	0.30%	114591.47	116891.02	0.24%	114046.48	116387.59	1.73%	**109288.52**	**109386.28**	0.03%	108330.10
B.u_s_lolo	1333.20	1191.70	1230.30	1173.50	1196.40	1174.02	1190.76	0.29%	1172.16	1183.08	0.09%	1170.19	1172.33	0.10%	**1143.25**	**1153.23**	0.27%	1128.10

Table 5.20 Makespan results for 2048x64 dataset instances.

Instance	min-min[77]	pCHC[117]		TPVNS[138]		GA+VNS			ACO+VNS			ACO(VNS)			GA(VNS)			LB
		Best	Avg	Best	Avg	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	Best	Avg	σ	
A.u_c_hihi	19552221.80	18110479.10	18218285.60	17795863.00	17801492.20	17930224.34	17933872.52	0.14%	17920127.75	17925663.51	0.21%	17895612.14	17900436.41	0.01%	**17241145.39**	**17253426.90**	0.05%	17141977.40
A.u_c_hilo	1873134.20	1748509.20	1760141.20	1727248.00	1736971.30	1736034.71	1740010.36	0.26%	1735017.36	1739016.48	0.20%	1733300.10	1733703.35	0.01%	**1673538.85**	**1676784.20**	0.23%	1664592.80
A.u_c_lohi	1924.70	1798.40	1804.90	1761.00	1770.60	1766.01	1770.02	0.19%	1762.53	1765.06	0.08%	1761.09	1761.82	0.02%	**1708.24**	**1711.09**	0.17%	1695.30
A.u_c_lolo	191.60	177.60	178.10	174.30	175.50	178.01	179.08	0.25%	177.12	178.01	0.16%	176.48	176.79	0.09%	**169.32**	**169.59**	0.24%	168.30
A.u_i_hihi	3248935.40	2506258.50	2546459.70	2478011.00	2500937.20	2492445.85	2556196.14	0.92%	2478079.39	2550203.45	1.04%	2470227.48	2547725.05	2.36%	**2434880.15**	**2451079.04**	0.26%	2366682.10
A.u_i_hilo	365828.60	272741.30	273876.30	274378.40	276000.10	275980.20	276763.30	0.20%	273187.20	274056.51	0.12%	271060.28	273342.21	0.60%	**269022.07**	**269887.09**	0.19%	260904.50
A.u_i_lohi	320.90	266.30	267.50	265.90	266.20	267.06	268.45	1.62%	266.04	267.23	1.39%	265.10	270.54	1.56%	**262.02**	**263.50**	0.19%	255.20
A.u_i_lolo	32.30	26.40	26.50	26.70	26.90	26.80	27.03	0.25%	26.51	26.86	0.33%	26.01	26.76	1.93%	**25.70**	**26.00**	0.24%	25.10
A.u_s_hihi	11245679.60	9756499.70	9821934.50	9524603.00	9601364.00	9516944.72	9584066.24	0.15%	9486944.72	9534920.93	0.11%	9310636.32	9506570.57	1.75%	**9291522.52**	**9317604.64**	0.15%	9050260.80
A.u_s_hilo	1042948.50	924094.90	937998.80	894695.30	909381.60	901335.20	909103.95	0.22%	893897.89	908484.01	0.22%	880835.16	910436.53	0.66%	**874008.82**	**879869.25**	0.20%	851399.90
A.u_s_lohi	1056.00	947.10	952.30	931.60	936.10	950.01	951.05	0.13%	948.01	949.09	0.12%	947.85	948.69	0.05%	**906.95**	**909.99**	0.17%	888.90
A.u_s_lolo	115.30	99.60	100.40	97.00	98.90	99.50	100.40	0.17%	97.05	99.00	0.25%	96.09	97.82	0.55%	**95.55**	**96.21**	0.21%	92.30
B.u_c_hihi	5564664.30	5290128.20	5300316.10	5209573.00	5219961.30	5220221.02	5230215.58	0.18%	5190033.71	5200359.73	0.19%	5183526.58	5185288.69	0.02%	**5006270.99**	**5013212.53**	0.11%	4975778.80
B.u_c_hilo	59352.80	55316.20	55343.10	53960.30	54001.50	53877.78	54801.88	0.22%	53434.14	54700.63	0.31%	53126.29	54414.88	0.13%	**52506.24**	**52579.58**	0.17%	52240.60
B.u_c_lohi	190842.40	177063.40	177612.40	175429.40	176981.20	175824.41	176400.03	0.08%	175502.36	176004.50	0.07%	175407.88	175793.58	0.13%	**173351.66**	**173399.65**	0.01%	167381.10
B.u_c_lolo	1927.70	1814.70	1818.30	1786.30	1791.00	1789.04	1795.09	0.44%	1786.10	1790.03	0.28%	1785.00	1785.92	0.02%	**1780.87**	**1781.43**	0.02%	1715.00
B.u_i_hihi	929295.80	770110.60	774993.00	765966.90	769121.10	769023.93	771990.87	0.23%	765811.16	771052.48	0.40%	765727.53	770771.60	0.32%	**762468.55**	**764872.14**	0.18%	735101.50
B.u_i_hilo	10318.40	7906.50	7932.90	7896.90	7910.10	7903.74	7990.60	1.18%	7899.81	7950.44	0.69%	7896.47	7920.60	0.17%	**7882.42**	**7907.56**	0.18%	7536.30
B.u_i_lohi	34071.00	26941.20	27207.30	27118.90	27900.40	27210.64	27895.77	0.23%	27111.15	27702.23	0.20%	27023.34	27637.86	0.34%	**26212.04**	**26539.76**	0.20%	25681.20
B.u_i_lolo	355.70	262.40	264.70	264.90	265.80	264.65	266.46	0.60%	263.37	265.98	0.87%	261.13	265.42	1.19%	**255.43**	**256.77**	0.19%	250.50
B.u_s_hihi	3293157.10	2910507.60	2923857.10	2865250.00	2876310.00	2891184.58	2910027.70	0.43%	2869958.73	2890179.77	0.46%	2867105.20	2885341.34	0.05%	**2745557.74**	**2753099.90**	0.19%	2710024.00
B.u_s_hilo	33445.40	29442.20	29518.60	28520.40	28731.20	28797.18	29257.77	0.26%	28513.85	29136.36	0.35%	28441.56	29053.08	0.99%	**27912.39**	**28001.94**	0.20%	27268.00
B.u_s_lohi	111237.40	98607.00	98758.30	94777.90	95101.40	95040.06	95428.05	0.22%	94634.51	95300.75	0.21%	94465.30	95243.41	0.65%	**93461.51**	**93683.73**	0.20%	90727.30
B.u_s_lolo	1163.80	1014.30	1019.70	995.80	1003.20	1012.12	1020.01	0.22%	994.26	1015.24	0.60%	990.35	1013.01	0.12%	**956.97**	**959.58**	0.18%	939.00

Table 5.21 Average improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) over some methods from the literature for the 1024x32 dataset.

Instance	GA+VNS			ACO+VNS			ACO(VNS)			GA(VNS)		
	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	min–min	pCHC	TPVNS
A.u_c_hihi	10.61	1.90	0.80	11.52	2.90	1.82	11.95	3.37	2.29	12.76	4.26	3.19
A.u_c_hilo	8.79	0.03	-0.33	9.17	0.45	0.09	9.85	1.20	0.84	13.15	4.82	4.47
A.u_c_lohi	8.62	1.54	0.05	8.85	1.79	0.30	11.57	4.72	3.28	12.76	6.00	4.58
A.u_c_lolo	6.72	3.25	1.25	7.84	4.42	2.45	11.36	8.06	6.16	11.46	8.16	6.27
A.u_i_hihi	18.37	0.88	0.45	19.01	1.66	1.24	19.83	2.66	2.24	20.13	3.02	2.60
A.u_i_hilo	23.24	0.07	0.29	23.45	0.34	0.56	24.34	1.50	1.72	25.08	2.46	2.67
A.u_i_lohi	20.40	-1.05	0.18	20.71	-0.66	0.58	21.63	0.50	1.72	23.30	2.63	3.82
A.u_i_lolo	18.45	-0.48	-0.10	18.98	0.17	0.56	21.08	2.76	3.14	22.12	4.04	4.41
A.u_s_hihi	13.01	1.22	0.15	13.31	1.56	0.49	14.97	3.44	2.39	16.96	5.70	4.68
A.u_s_hilo	10.18	2.43	0.05	10.39	2.66	0.29	10.90	3.21	0.86	13.28	5.80	3.51
A.u_s_lohi	13.56	0.49	0.11	14.01	1.01	0.63	15.03	2.18	1.80	17.72	5.28	4.92
A.u_s_lolo	10.78	0.83	-1.91	12.48	2.72	0.03	13.13	3.44	0.78	16.27	6.94	4.37
B.u_c_hihi	6.83	-0.80	-0.80	7.64	0.07	0.07	8.99	1.54	1.54	10.43	3.09	3.09
B.u_c_hilo	7.74	-0.22	0.12	7.82	-0.12	0.21	9.51	1.71	2.04	10.96	3.28	3.60
B.u_c_lohi	6.76	0.83	0.07	7.32	1.42	0.67	9.46	3.70	2.97	9.62	3.86	3.13
B.u_c_lolo	9.57	2.28	0.02	9.80	2.54	0.27	11.87	4.77	2.56	12.51	5.47	3.27
B.u_i_hihi	24.69	2.39	-0.09	24.79	2.52	0.05	24.93	2.70	0.23	26.03	4.13	1.69
B.u_i_hilo	9.55	0.08	1.68	10.42	1.04	2.62	11.85	2.62	4.18	13.08	3.98	5.52
B.u_i_lohi	11.20	-0.27	-0.56	11.83	0.45	0.16	12.67	1.39	1.10	14.74	3.73	3.45
B.u_i_lolo	14.18	1.05	-0.13	14.51	1.44	0.26	15.95	3.10	1.94	17.00	4.31	3.17
B.u_s_hihi	14.56	1.18	0.07	15.03	1.73	0.63	16.10	2.96	1.87	18.13	5.31	4.25
B.u_s_hilo	10.28	0.02	1.09	11.73	1.64	2.69	13.89	4.04	5.07	14.71	4.96	5.97
B.u_s_lohi	13.35	0.47	-0.43	13.81	1.00	0.11	14.18	1.43	0.54	19.35	7.35	6.52
B.u_s_lolo	10.68	3.21	0.47	11.26	3.84	1.11	12.07	4.71	2.01	13.50	6.26	3.61
Avg	12.59	0.89	0.11	13.15	1.52	0.75	14.46	2.99	2.22	16.04	4.78	4.03

Table 5.22 Average improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) over some methods from the literature for the 2048x64 dataset.

Instance	GA+VNS			ACO+VNS			ACO(VNS)			GA(VNS)		
	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	min-min	pCHC	TPVNS	min–min	pCHC	TPVNS
A.u_c_hihi	8.28	1.56	-0.74	8.32	1.61	-0.70	8.45	1.74	-0.56	11.76	5.30	3.08
A.u_c_hilo	7.11	1.14	-0.17	7.16	1.20	-0.12	7.44	1.50	0.19	10.48	4.74	3.47
A.u_c_lohi	8.04	1.93	0.03	8.29	2.21	0.31	8.46	2.39	0.50	11.10	5.20	3.36
A.u_c_lolo	6.53	-0.55	-2.04	7.09	0.05	-1.43	7.73	0.74	-0.73	11.49	4.78	3.37
A.u_i_hihi	21.32	-0.38	-2.21	21.51	-0.15	-1.97	21.58	-0.05	-1.87	24.56	3.75	1.99
A.u_i_hilo	24.35	-1.05	-0.28	25.09	-0.07	0.70	25.28	0.20	0.96	26.23	1.46	2.21
A.u_i_lohi	16.34	-0.36	-0.85	16.72	0.10	-0.39	15.69	-1.13	-1.63	17.89	1.50	1.01
A.u_i_lolo	16.32	-2.00	-0.48	16.84	-1.36	0.15	17.15	-0.98	0.52	19.50	1.89	3.35
A.u_s_hihi	14.78	2.42	0.18	15.21	2.92	0.69	15.46	3.21	0.99	17.15	5.13	2.96
A.u_s_hilo	12.83	3.08	0.03	12.89	3.15	0.10	12.71	2.94	-0.12	15.64	6.20	3.25
A.u_s_lohi	9.94	0.13	-1.60	10.12	0.34	-1.39	10.16	0.38	-1.35	13.83	4.44	2.79
A.u_s_lolo	12.92	0.00	-1.52	14.14	1.39	-0.10	15.16	2.57	1.10	16.56	4.17	2.72
B.u_c_hihi	6.01	1.32	-0.20	6.55	1.89	0.38	6.82	2.17	0.66	9.91	5.42	3.96
B.u_c_hilo	7.67	0.98	-1.48	7.84	1.16	-1.29	8.32	1.68	-0.77	11.41	4.99	2.63
B.u_c_lohi	7.57	0.68	0.33	7.77	0.91	0.55	7.89	1.02	0.67	9.14	2.37	2.02
B.u_c_lolo	6.88	1.28	-0.23	7.14	1.55	0.05	7.36	1.78	0.28	7.59	2.03	0.53
B.u_i_hihi	16.93	0.39	-0.37	17.03	0.51	-0.25	17.06	0.54	-0.21	17.69	1.31	0.55
B.u_i_hilo	22.56	-0.73	-1.02	22.95	-0.22	-0.51	23.24	0.16	-0.13	23.36	0.32	0.03
B.u_i_lohi	18.12	-2.53	0.02	18.69	-1.82	0.71	18.88	-1.58	0.94	22.10	2.45	4.88
B.u_i_lolo	25.09	-0.66	-0.25	25.22	-0.48	-0.07	25.38	-0.27	0.14	27.81	3.00	3.40
B.u_s_hihi	11.63	0.47	-1.17	12.24	1.15	-0.48	12.38	1.32	-0.31	16.40	5.84	4.28
B.u_s_hilo	12.52	0.88	-1.83	12.88	1.29	-1.41	13.13	1.58	-1.12	16.28	5.14	2.54
B.u_s_lohi	14.21	3.37	-0.34	14.33	3.50	-0.21	14.38	3.56	-0.15	15.78	5.14	1.49
B.u_s_lolo	12.36	-0.03	-1.68	12.77	0.44	-1.20	12.96	0.66	-0.98	17.55	5.90	4.35
Avg	13.35	0.47	-0.74	13.70	0.89	-0.33	13.88	1.09	-0.12	16.30	3.85	2.68

Table 5.23 Average improvement percentages and statistical analysis of GA(VNS) over other methods for the 1024x32 dataset.

Instance	GA+VNS		ACO+VNS		ACO(VNS)	
	improvement	p-value	improvement	p-value	improvement	p-value
A.u_c_hihi	2.41	$<10^{-5}$	1.40	$<10^{-5}$	0.92	$<10^{-5}$
A.u_c_hilo	4.78	$<10^{-5}$	4.38	$<10^{-5}$	3.66	$<10^{-5}$
A.u_c_lohi	4.53	$<10^{-5}$	4.29	$<10^{-5}$	1.34	$<10^{-5}$
A.u_c_lolo	5.08	$<10^{-5}$	3.92	$<10^{-5}$	0.11	$<10^{-5}$
A.u_i_hihi	2.16	$<10^{-5}$	1.38	$<10^{-5}$	0.37	$<10^{-5}$
A.u_i_hilo	2.39	$<10^{-5}$	2.13	$<10^{-5}$	0.97	$<10^{-5}$
A.u_i_lohi	3.64	$<10^{-5}$	3.26	$<10^{-5}$	2.13	$<10^{-5}$
A.u_i_lolo	4.50	$<10^{-5}$	3.87	$<10^{-5}$	1.31	$<10^{-5}$
A.u_s_hihi	4.54	$<10^{-5}$	4.21	$<10^{-5}$	2.34	$<10^{-5}$
A.u_s_hilo	3.46	$<10^{-5}$	3.23	$<10^{-5}$	2.68	$<10^{-5}$
A.u_s_lohi	4.82	$<10^{-5}$	4.32	$<10^{-5}$	3.17	$<10^{-5}$
A.u_s_lolo	6.16	$<10^{-5}$	4.34	$<10^{-5}$	3.62	$<10^{-5}$
B.u_c_hihi	3.86	$<10^{-5}$	3.02	$<10^{-5}$	1.58	$<10^{-5}$
B.u_c_hilo	3.49	$<10^{-5}$	3.40	$<10^{-5}$	1.60	$<10^{-5}$
B.u_c_lohi	3.06	$<10^{-5}$	2.48	$<10^{-5}$	0.17	$<10^{-5}$
B.u_c_lolo	3.26	$<10^{-5}$	3.01	$<10^{-5}$	0.73	$<10^{-5}$
B.u_i_hihi	1.78	$<10^{-5}$	1.65	$<10^{-5}$	1.47	$<10^{-5}$
B.u_i_hilo	3.91	$<10^{-5}$	2.97	$<10^{-5}$	1.40	$<10^{-5}$
B.u_i_lohi	3.99	$<10^{-5}$	3.29	$<10^{-5}$	2.37	$<10^{-5}$
B.u_i_lolo	3.29	$<10^{-5}$	2.91	$<10^{-5}$	1.25	$<10^{-5}$
B.u_s_hihi	4.18	$<10^{-5}$	3.64	$<10^{-5}$	2.42	$<10^{-5}$
B.u_s_hilo	4.93	$<10^{-5}$	3.37	$<10^{-5}$	0.96	$<10^{-5}$
B.u_s_lohi	6.92	$<10^{-5}$	6.42	$<10^{-5}$	6.02	$<10^{-5}$
B.u_s_lolo	3.15	$<10^{-5}$	2.52	$<10^{-5}$	1.63	$<10^{-5}$
Avg	3.93		3.31		1.84	

Table 5.24 Average improvement percentages and statistical analysis of GA(VNS) over other methods for the 2048x64 dataset.

Instance	GA+VNS		ACO+VNS		ACO(VNS)	
	improvement	p-value	improvement	p-value	improvement	p-value
A.u_c_hihi	3.79	$<10^{-5}$	3.75	$<10^{-5}$	3.61	$<10^{-5}$
A.u_c_hilo	3.63	$<10^{-5}$	3.58	$<10^{-5}$	3.28	$<10^{-5}$
A.u_c_lohi	3.33	$<10^{-5}$	3.06	$<10^{-5}$	2.88	$<10^{-5}$
A.u_c_lolo	5.30	$<10^{-5}$	4.73	$<10^{-5}$	4.07	$<10^{-5}$
A.u_i_hihi	4.11	$<10^{-5}$	3.89	$<10^{-5}$	3.79	$<10^{-5}$
A.u_i_hilo	2.48	$<10^{-5}$	1.52	$<10^{-5}$	1.26	$<10^{-5}$
A.u_i_lohi	1.84	$<10^{-5}$	1.40	$<10^{-5}$	2.60	$<10^{-5}$
A.u_i_lolo	3.81	$<10^{-5}$	3.20	$<10^{-5}$	2.84	$<10^{-5}$
A.u_s_hihi	2.78	$<10^{-5}$	2.28	$<10^{-5}$	1.99	$<10^{-5}$
A.u_s_hilo	3.22	$<10^{-5}$	3.15	$<10^{-5}$	3.36	$<10^{-5}$
A.u_s_lohi	4.32	$<10^{-5}$	4.12	$<10^{-5}$	4.08	$<10^{-5}$
A.u_s_lolo	4.17	$<10^{-5}$	2.82	$<10^{-5}$	1.64	$<10^{-5}$
B.u_c_hihi	4.15	$<10^{-5}$	3.60	$<10^{-5}$	3.32	$<10^{-5}$
B.u_c_hilo	4.06	$<10^{-5}$	3.88	$<10^{-5}$	3.37	$<10^{-5}$
B.u_c_lohi	1.70	$<10^{-5}$	1.48	$<10^{-5}$	1.36	$<10^{-5}$
B.u_c_lolo	0.76	$<10^{-5}$	0.48	$<10^{-5}$	0.25	$<10^{-5}$
B.u_i_hihi	0.92	$<10^{-5}$	0.80	$<10^{-5}$	0.77	$<10^{-5}$
B.u_i_hilo	1.04	$<10^{-5}$	0.54	$<10^{-5}$	0.16	0.00009
B.u_i_lohi	4.86	$<10^{-5}$	4.20	$<10^{-5}$	3.97	$<10^{-5}$
B.u_i_lolo	3.64	$<10^{-5}$	3.46	$<10^{-5}$	3.26	$<10^{-5}$
B.u_s_hihi	5.39	$<10^{-5}$	4.74	$<10^{-5}$	4.58	$<10^{-5}$
B.u_s_hilo	4.29	$<10^{-5}$	3.89	$<10^{-5}$	3.62	$<10^{-5}$
B.u_s_lohi	1.83	$<10^{-5}$	1.70	$<10^{-5}$	1.64	$<10^{-5}$
B.u_s_lolo	5.92	$<10^{-5}$	5.48	$<10^{-5}$	5.27	$<10^{-5}$
Avg	3.39		2.99		2.79	

In Tables 5.19 and 5.20, the last column represents the Lower Bound (LB) values of each problem instance as reported in [117]. Tables 5.25 and 5.26 summarize the gap values between the average makespan results for GA+VNS, ACO+VNS, ACO(VNS), GA(VNS) and selected algorithms from the literature and its corresponding lower bounds for the 1024x32 and 2048x64 dataset instances. For 1024x32 instances, GA(VNS) achieved the smallest average gaps to the lower bound at 1.26, while min-min, pCHC, TPVNS,

GA+VNS, ACO+VNS and ACO(VNS) achieved 21.00, 6.38, 5.53 , 5.42, 4.74 and 3.18, respectively. For 2048x64 instances, GA(VNS) also achieved the smallest average gap with the lower bound at 2.64, while min-min, pCHC, TPVNS, GA+VNS, ACO+VNS and ACO(VNS) achieved 23.21, 6.77, 5.46, 6.25, 5.81 and 5.59, respectively. This indicates that the quality of the solutions found by GA(VNS) is very high compared to the others.

Table 5.25 The gaps for the average makespan of the GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) methods and the selected algorithms from the literature and their corresponding lower bounds for the 1024x32 dataset.

Instance	min–min	pCHC	TPVNS	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
A.u_c_hihi	15.73	5.46	4.29	3.45	2.39	1.90	0.96
A.u_c_hilo	15.61	5.48	5.10	5.45	5.01	4.22	0.40
A.u_c_lohi	15.46	7.16	5.56	5.51	5.25	2.10	0.73
A.u_c_lolo	13.57	9.50	7.29	5.95	4.67	0.68	0.56
A.u_i_hihi	27.05	4.63	4.18	3.71	2.89	1.85	1.47
A.u_i_hilo	35.21	3.86	4.09	3.78	3.50	2.30	1.31
A.u_i_lohi	32.04	4.01	5.30	5.11	4.70	3.49	1.28
A.u_i_lolo	30.00	5.51	5.92	6.02	5.33	2.60	1.25
A.u_s_hihi	22.26	7.67	6.51	6.36	5.99	3.96	1.53
A.u_s_hilo	17.09	7.79	5.23	5.17	4.92	4.33	1.53
A.u_s_lohi	23.02	6.86	6.45	6.34	5.78	4.53	1.22
A.u_s_lolo	20.48	8.40	5.48	7.50	5.45	4.67	0.87
B.u_c_hihi	12.16	3.67	3.67	4.50	3.59	2.07	0.46
B.u_c_hilo	13.13	4.15	4.51	4.38	4.28	2.37	0.74
B.u_c_lohi	11.60	4.92	4.13	4.05	3.43	1.04	0.87
B.u_c_lolo	14.84	6.28	3.86	3.85	3.58	1.21	0.47
B.u_i_hihi	38.12	6.57	3.93	4.02	3.88	3.69	2.17
B.u_i_hilo	17.15	6.05	7.77	5.97	4.95	3.27	1.83
B.u_i_lohi	18.89	5.30	4.99	5.58	4.82	3.83	1.37
B.u_i_lolo	22.54	6.28	5.03	5.16	4.75	2.99	1.70
B.u_s_hihi	24.82	7.92	6.72	6.64	6.05	4.72	2.19
B.u_s_hilo	19.86	7.56	8.73	7.54	5.80	3.22	2.23
B.u_s_lohi	25.20	8.99	8.02	8.48	7.90	7.44	0.97
B.u_s_lolo	18.18	9.06	6.05	5.55	4.87	3.92	2.23
Avg	21.00	6.38	5.53	5.42	4.74	3.18	1.26

Table 5.26 The gaps for the average makespan of the GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) methods and the selected algorithms from the literature and their corresponding lower bounds for the 2048x64 dataset.

Instance	min–min	pCHC	TPVNS	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
A.u_c_hihi	14.06	6.28	3.85	4.62	4.57	4.42	0.65
A.u_c_hilo	12.53	5.74	4.35	4.53	4.47	4.15	0.73
A.u_c_lohi	13.53	6.46	4.44	4.41	4.11	3.92	0.93
A.u_c_lolo	13.84	5.82	4.28	6.41	5.77	5.04	0.77
A.u_i_hihi	37.28	7.60	5.67	8.01	7.75	7.65	3.57
A.u_i_hilo	40.22	4.97	5.79	6.08	5.04	4.77	3.44
A.u_i_lohi	25.74	4.82	4.31	5.19	4.71	6.01	3.25
A.u_i_lolo	28.69	5.58	7.17	7.69	7.01	6.61	3.59
A.u_s_hihi	24.26	8.53	6.09	5.90	5.36	5.04	2.95
A.u_s_hilo	22.50	10.17	6.81	6.78	6.70	6.93	3.34
A.u_s_lohi	18.80	7.13	5.31	6.99	6.77	6.73	2.37
A.u_s_lolo	24.92	8.78	7.15	8.78	7.26	5.98	4.24
B.u_c_hihi	11.84	6.52	4.91	5.11	4.51	4.21	0.75
B.u_c_hilo	13.61	5.94	3.37	4.90	4.71	4.16	0.65
B.u_c_lohi	14.02	6.11	5.74	5.39	5.15	5.03	3.60
B.u_c_lolo	12.40	6.02	4.43	4.67	4.37	4.14	3.87
B.u_i_hihi	26.42	5.43	4.63	5.02	4.89	4.85	4.05
B.u_i_hilo	36.92	5.26	4.96	6.03	5.50	5.10	4.93
B.u_i_lohi	32.67	5.94	8.64	8.62	7.87	7.62	3.34
B.u_i_lolo	42.00	5.67	6.11	6.37	6.18	5.95	2.50
B.u_s_hihi	21.52	7.89	6.14	7.38	6.65	6.47	1.59
B.u_s_hilo	22.65	8.25	5.37	7.30	6.85	6.55	2.69
B.u_s_lohi	22.61	8.85	4.82	5.18	5.04	4.98	3.26
B.u_s_lolo	23.94	8.59	6.84	8.63	8.12	7.88	2.19
Avg	23.21	6.77	5.46	6.25	5.81	5.59	2.64

5.6 Results summary for Braun *et al.* [19] and Nesmachnow *et al.* [118] datasets

Table 5.27 and Fig. 5.6 summarize the average improvements of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) over the deterministic heuristic min-min algorithm for the problem instances of Braun *et al.* and Nesmachnow *et al.*, where the improvement

Table 5.27 GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) average improvement percentages with respect to the min-min heuristic based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

Consistency	512x16			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	8.90	9.21	9.51	9.89
inconsistent	11.80	11.95	12.15	12.30
semiconsistent	11.71	12.10	12.26	13.21
	A-1024x32			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	8.68	8.68	12.53	12.53
inconsistent	20.11	20.11	22.65	22.65
semiconsistent	11.88	11.88	16.06	16.06
	B-1024x32			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	7.73	8.15	9.96	10.88
inconsistent	14.90	15.39	16.35	17.71
semiconsistent	12.22	12.96	14.06	16.42
	A-2048x64			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	7.49	7.72	8.02	11.21
inconsistent	19.58	20.04	19.93	22.04
semiconsistent	12.62	13.09	13.37	15.79
	B-2048x64			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	7.03	7.33	7.59	9.51
inconsistent	20.68	20.97	21.14	22.74
semiconsistent	12.68	13.05	13.21	16.50

percentages are categorized based on consistency. The average improvement percentages of ACO(VNS) with respect to the min-min heuristic were always above 7% and 12% for the consistent and semi-consistent instances, respectively, while GA(VNS) was more accurate showing average improvement percentages of no less than 9% and 13%, respectively. For the inconsistent instances, both algorithms achieved average improvements above 12% for the 512x16 dataset. However, this percentage increased significantly to greater than 21% for larger problem instances. On the other hand, the average improvement percentages of GA+VNS with respect to the min-min heuristic were always above 7% and 11% for the consistent and semi-consistent instances, respectively, while ACO+VNS was a slightly more accurate showing average improvement percentages of no less than 7% and 11%, respectively. For the inconsistent instances, both algorithms achieved average improvements above 11% for the 512x16 dataset. However, this percentage increased significantly to greater than 20% for larger problem instances.

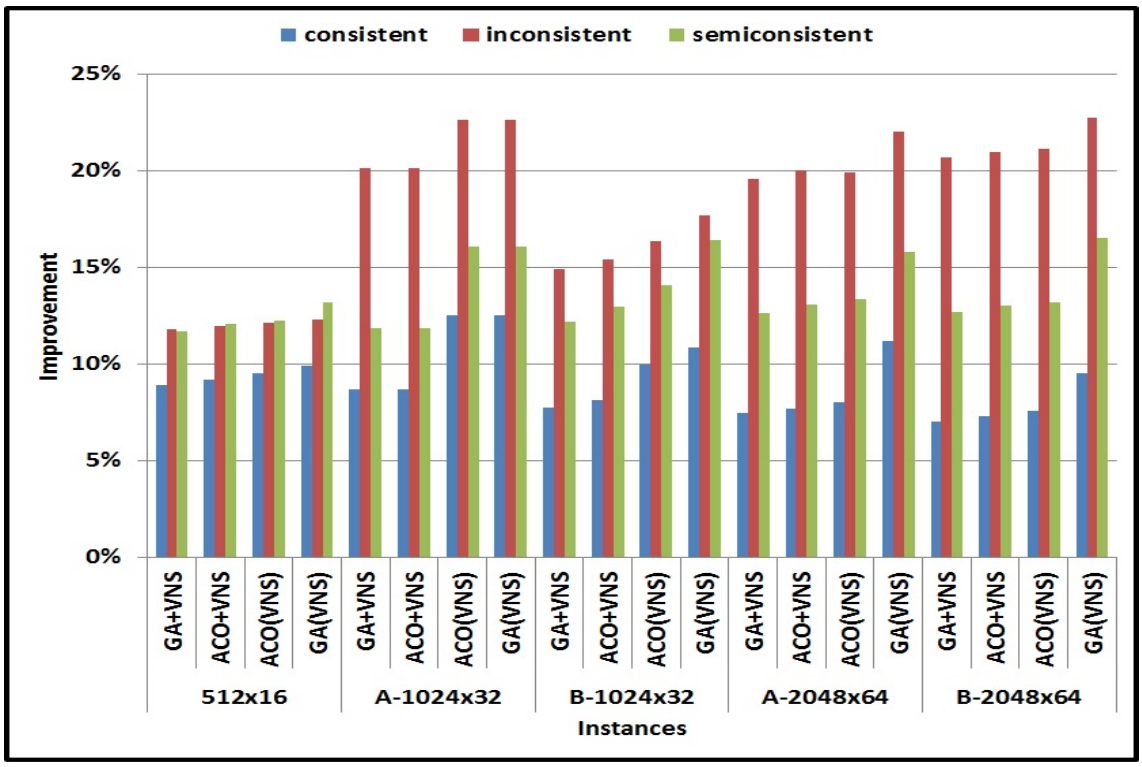

Fig. 5.6 The graphical average improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) methods with respect to the min-min heuristic based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

Table 5.28 reports the average gap percentage of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) with regard to the lower bound for the datasets of Braun *et al.* and Nesmachnow *et al.* based on the consistency. For consistent and inconsistent instances, ACO(VNS), ACO+VNS and GA+VNS showed stable behaviour. However, for the semi-consistent instances, it may be noted that they have a high average gap percentages for all datasets. GA(VNS) showed stable behaviour regarding all types of consistency.

Fig. 5.7 demonstrates the average improvement percentages of GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) over the *ad hoc* min-min method and the average percentages of the corresponding gap to the lower bounds of the Braun *et al.* and Nesmachnow *et al.* datasets. It may be noted that ACO(VNS) and GA(VNS) are capable of achieving high-quality mappings which are very close to the lower bounds. However, as the dataset size grows, the average gap percentage for ACO(VNS) increases compared to GA(VNS), indicating greater stability. On the other hand, it may be seen that GA+VNS and ACO+VNS are also capable of achieving good-quality mappings which are relatively close to the lower bounds. However, as the dataset size grows, the average gap percentage for them increases compared to GA(VNS).

Table 5.28 GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) average gap percentages with respect to the lower bound based on the consistency: Braun *et al.* and Nesmachnow *et al.* datasets.

Consistency	512x16			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	1.58	1.23	0.89	0.46
inconsistent	1.35	1.19	0.96	0.78
semiconsistent	2.65	2.20	2.00	0.88
	A-1024x32			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	5.09	4.33	2.22	0.66
inconsistent	4.66	4.10	2.56	1.33
semiconsistent	6.34	5.54	4.37	1.29
	B-1024x32			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	4.19	3.72	1.68	0.64
inconsistent	5.18	4.60	3.45	1.77
semiconsistent	7.05	6.16	4.82	1.91
	A-2048x64			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	4.99	4.73	4.39	0.77
inconsistent	6.74	6.13	6.26	3.46
semiconsistent	7.11	6.52	6.17	3.23
	B-2048x64			
	GA+VNS	ACO+VNS	ACO(VNS)	GA(VNS)
consistent	5.02	4.69	4.38	2.22
inconsistent	6.51	6.11	5.88	3.71
semiconsistent	7.12	6.66	6.47	2.43

5.7 Summary

In this chapter, the application of various hybrid meta-heuristic algorithms for solving the independent static job scheduling problem in grid computing in terms of minimising the makespan was discussed. To evaluate the performance of the proposed methods, the ETC model has been used. Three different well-known datasets have been used to perform several experiments. The experimental results show that GA(VNS) achieved results that were significantly better than GA+VNS, ACO+VNS, ACO(VNS) and other selected approaches from the literature for all three benchmarks used in terms of minimising the makespan; therefore, we can claim that it represents the new state-of-the-art sequential hybrid algorithm for job scheduling in grid computing. With very low standard deviation values, it should be expected that GA(VNS) can find high-quality schedules in any single run. Moreover, GA(VNS) achieved results that show the smallest gap with the lower bound in all the problem instances examined in this chapter. ACO(VNS) was almost the

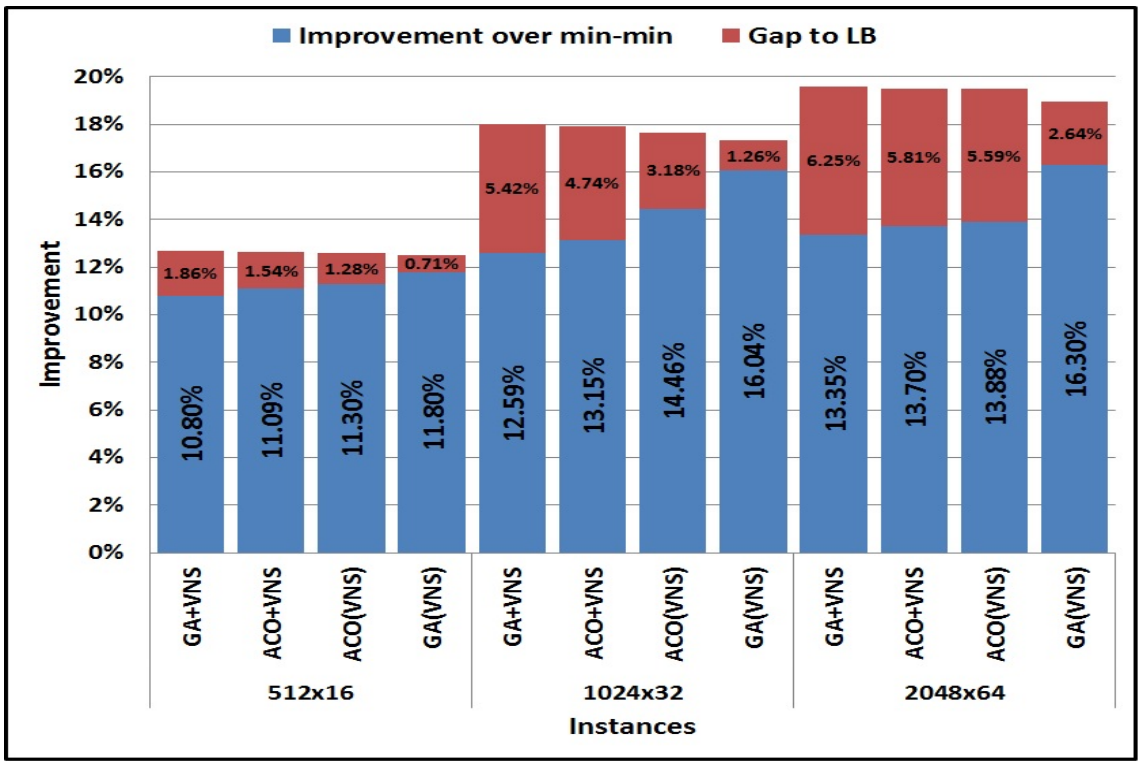

Fig. 5.7 GA+VNS, ACO+VNS, ACO(VNS) and GA(VNS) average improvement percentages over the min-min heuristic and the corresponding gap to LB: Braun *et al.* and Nesmachnow *et al.* datasets.

second-best algorithm in terms of makespan results; however, it needed a longer time to construct high-quality solutions. For relatively small problem instances, the result for ACO(VNS) were very close to the ones achieved by GA(VNS); however, as the dataset size increases, the quality of the solutions found by ACO(VNS) decreases, which means that longer times will be needed to improve the results.

Chapter 6

Hybrid Meta-Heuristics for Dynamic Job Scheduling in Grid Computing

The previous chapter discussed the use of hybrid meta-heuristic algorithms for the static independent job scheduling in grid computing. This chapter introduces the application of hybrid meta-heuristic algorithms for solving the dynamic job scheduling problem in grid computing in terms of minimising the makespan. We will consider the version of the dynamic job scheduling problem in grid computing in which blocks of independent jobs arrive to the grid system at different arrival time. To solve this problem, the rescheduling strategy, which involves several calls of the job scheduler at various intervals of time, is employed. The dynamic simulation model and how to apply rescheduling are explained in the following sections. Moreover, the chapter studies the performance of the proposed meta-heuristic algorithms by using a benchmark which has been especially created for the dynamic job scheduling problem. Finally, some conclusions about the effect of using rescheduling are drawn.

6.1 Dynamic job scheduling simulation model

The dynamic version of job scheduling in grid computing involves the arrival of several groups of jobs with various sizes to the grid system at different times, which is a typical situation in dynamic heterogeneous computing environments, where not all jobs are available in advance, instead they arrive during the processing of previously submitted jobs.

A description of the dynamic job scheduling problem using the rescheduling strategy can be formulated as follows:

1. A set G of k groups, where $G = \{g_0, g_1, ..., g_{k-1}\}$. Each group $g_l = \{j_0, j_1, ..., j_{n_{l-1}}\}$, where $l \in$ [0, k-1] and n_l is the number of independent jobs in group g_l, which

varies from one group to another, is to be assigned to grid resources. Any job can be handled by any resource. However, these jobs are non-pre-emptive, i.e., each job should be executed entirely by one resource only. Unlike the static version of the problem in which all groups arrive at the same time, each group arrives to the system in a different arrival time.

2. A set of m heterogeneous resources $H = \{r_0, r_1, ..., r_{m-1}\}$ to be used for processing the independent jobs.

3. A processing time function ETC : $g_l \times \mathrm{H} \rightarrow \mathrm{R+}$, where $l \in$ [0,k-1] and $ETC[j][r]$ denotes the estimated required time for processing job j by resource r.

4. The goal of dynamic job scheduling in grid computing is to find a mapping of the submitted jobs onto the available resources (a function f: $g_l \rightarrow \mathrm{H}$, $l \in$ [0,k-1]) that minimizes the makespan, which represents the finishing time of the latest job and can be computed as follows:

$$makespan = min_{s \in S} max_{j \in g_l}(Finish_j), l \in [0, k-1] \tag{6.1}$$

where S is the set of all possible solutions and $Finish_j$ represents the time by which job j will be completed [85].

The dynamic simulation model assumes that the grid system receives each group of jobs at different arrival times by generating a random time in the range [0.5 x *Pmakespan*, *Pmakespan*], where *Pmakespan* is the makespan result of the previous group. When a new group of jobs arrives to the grid system for processing, a rescheduling operation is activated, which involves scheduling the jobs in the arrived group and those jobs already submitted but have not began their processing. Fig. 6.1 shows graphically an example of how to perform rescheduling in dynamic job scheduling. At t_{res}, a rescheduling operation is activated, which aims at assigning the new arrived jobs and all the unprocessed jobs, i.e. $[j_{19}, j_{23}, j_{22}, j_{10}, j_{12}, j_{21}, j_{18}, j_{20}]$, which have not been processed at that time, to the available resources.

The arrival of jobs to the grid system at runtime makes the job scheduling problem dynamic. One way to solve this dynamic problem is to schedule each group separately. Alternatively, a reschedule of the unprocessed jobs at t_{res} provides an opportunity to reassign some jobs to different resources, and hence, minimize the overall makespan of the solution.

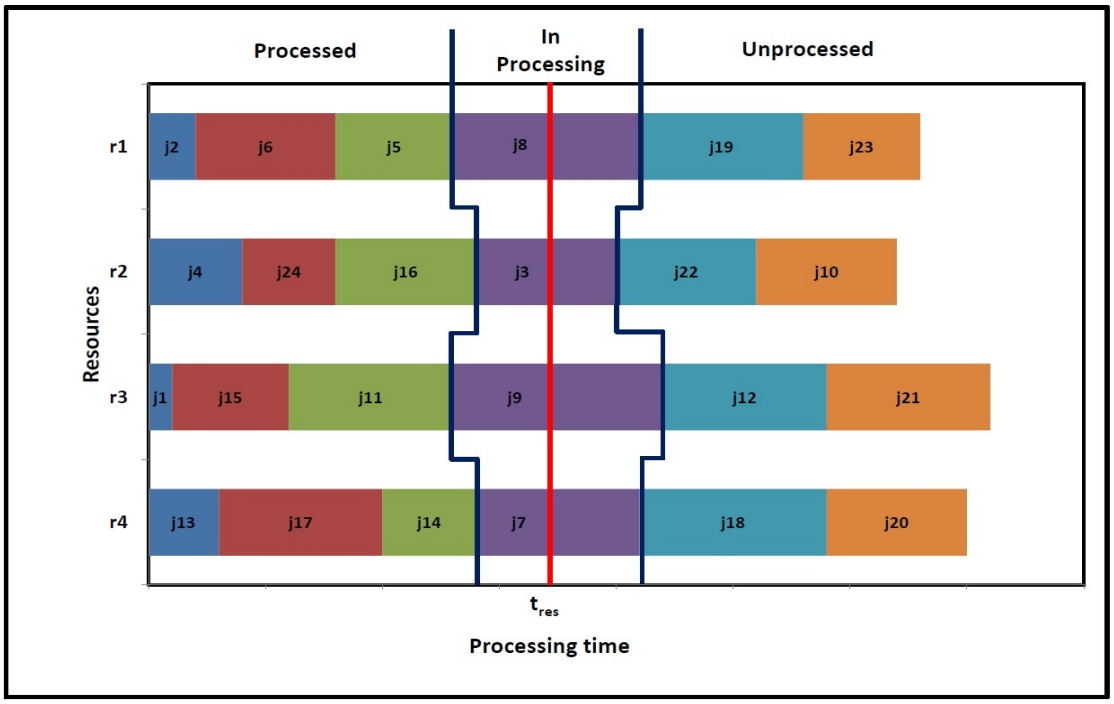

Fig. 6.1 An example of rescheduling in dynamic job scheduling.

6.2 Rescheduling simulator for dynamic job scheduling in grid computing

To implement the rescheduling operation, a simple rescheduling simulator for dynamic job scheduling in grid computing was designed as illustrated in Algorithm 6.1. This simulator provides the necessary steps required for performing the scheduling and rescheduling in heterogeneous dynamic environments such as grid computing. These steps include the simulation of the arrival of the groups of jobs, the activation of the scheduling/rescheduling algorithm, and linking the solutions resulted from applying the scheduling/rescheduling algorithm.

The simulator performs $k-1$ rescheduling operations, where k is the total number of groups. In each iteration, $no\_jobs + no\_unp\_jobs$ jobs are scheduled, where no_jobs is the number of jobs in the arrived group l and no_unp_jobs is the number unprocessed jobs when a rescheduling occurs. It is worth noting that in the first round $no\_unp\_jobs = 0$ as the grid system has not received any job yet. The scheduling is done using any job scheduling algorithm (line 7 in Algorithm 6.1), which could be heuristic, meta-heuristic or hybrid, that has been modified to perform rescheduling. The makespan of the solution, $Pmakespan$, is then calculated. The simulator uses it to generate the next rescheduling time, t_{res}, by randomly select a time from the range [0.5 x $Pmakespan$, $Pmakespan$]. Finally,

Algorithm 6.1: Dynamic job scheduling scenario using rescheduling

```
1  Let k be the total number of groups;
2  no_unp_jobs ← 0;
3  for (l=0 to k-1) do
4      no_jobs ← the number of jobs for the arrived group l;
5      no_jobs ← no_jobs + no_unp_jobs;
6      G ← ETC of the jobs in the arrived group l and unprocessed jobs;
7      schedule G;
8      Pmakespan ← makespan of the schedule G;
9      no_unp_jobs ← 0;
10     t_res ← rand(0.5 × Pmakespan, Pmakespan);
11     for (each resource) do
12         for (each unprocessed job at time t_res) do
13             no_unp_jobs++;
14         end
15     end
16 end
```

the simulator counts the number of unprocessed jobs, the jobs after t_{res}, which will be rescheduled in the next iteration.

6.3 Rescheduling-based methods

To solve the dynamic version of job scheduling and to implement the rescheduling operation, some modifications were added to the hybrid meta-heuristic schedulers described in Chapter 4. These modifications include:

1. A new class which reads the ETC values of the new arrived jobs and the unprocessed jobs.

2. After every rescheduling activation, a modification to the previous solution occurs. Therefore, a new class which saves the updated solution is required. This class keeps updating the solution until processing all groups, and then generates the final solution.

3. A new class that determines which jobs were processed, in processing, and unprocessed. These information are important for performing the next rescheduling operation.

4. The solution representation, in which the resource-based representation is used for all algorithms since the number of jobs is variable due to the dynamic nature of the problem.

5. For any GA-based scheduler, the CX crossover operator is used, which was the best permutation-based crossover operator, since the 2P operator is not applicable for the resource-based representation.

The above modifications were also applied on the min-min algorithm which was selected to study the performance of the proposed methods.

6.4 Experimental analysis

This section presents the experimental evaluation of applying the different hybrid meta-heuristic algorithms for solving the dynamic version of job scheduling in grid computing using the rescheduling strategy.

A new benchmark is created for this purpose, which consists of 12 problem instances to simulates the different job and resource heterogeneity cases. The range-based method [8], with the parameters described in Bruan *et. al.* [19], was used to generate this benchmark mainly because it represents the classical model to study the performance of scheduling algorithms in grid computing. Similar to Braun *et al.* dataset, the same notation was used to describe each problem instance in the new benchmark; however, to distinguish the new dynamic dataset instances from the classical Braun *et al.* dataset, each instance is preceded by the letter d. Like Braun *et al.* dataset, each instance has 512 jobs and 16 resources. However, the 512 jobs of each problem instance were divided into ten groups, which is the number of groups k, of various sizes, namely $\{62, 30, 59, 16, 80, 76, 100, 21, 14, 54\}$.

To study the effects of applying rescheduling, each algorithm has two versions, namely without rescheduling and with rescheduling. Any algorithm applies rescheduling will be preceded by r, for example rGA(VNS) means the algorithm GA(VNS) that uses a rescheduling technique, while GA(VNS) means the same algorithm without rescheduling. The algorithms without rescheduling schedule each group separately, while the algorithms with rescheduling follow the scenario described in Algorithm 6.1.

6.4.1 Parameter tuning

As with static version of the problem, a fix set of parameters was selected from the literature for each of the proposed algorithms to perform parameter tuning process. A number of instances with diverse characteristics were used to carry out the parameter tuning experiments. For the proposed VNS, the examined parameter was the order of neighbourhood structures only. Population size, α, β, pheromone evaporation rate (ρ) were among the tested parameters for the proposed hybrid rACO+VNS and rACO(VNS). On the other hand, the following parameters were examined for the proposed hybrid rGA+VNS and rGA(VNS): population size, crossover type, mutation type, crossover probability and

mutation probability. To select the best parameter values, each algorithm was executed 10 times for each problem instance and for each parameter, and their average was reported.

6.4.1.1 Parameter tuning for VNS

One of the main advantages of VNS is that it does not need many parameters. The stopping condition is the maximum number of iterations, which was set to 5. As mentioned earlier, the order of neighbourhood structures will be the main parameter that will be examined, as the forward VNS version is used in this study which means that VNS starts with k=1 and then increases k by one if no improvement is found; otherwise, set k=1. Since we have four different structures, we then have 24 possible combinations. Table 6.1 illustrates the effects of changing the order of neighbourhood structures based on different instances with different characteristics. The tables show that case 24 was the best order recorded in almost all the tested cases.

6.4.1.2 Parameter tuning for rACO+VNS and rACO(VNS)

Three parameters have been examined for the hybrid rACO+VNS and rACO(VNS) which are the population size, the values of α and β and the value of ρ. The population size is set to 2 due to the attempt to reduce the computational time needed to construct solutions by ants and increase the number of generations. Various studies have suggested optimal values for α and β which vary between 1 and 10, while the suggested values for ρ were between 0.5 and 0.7 [135] [147] [60]. Therefore, three values have been used for α and β, which are 1, 5 and 10. The results indicate that α=10 and β=1 represented the best combination, as shown in Table 6.2; similarly, two values were used for ρ, which are 0.5 and 0.7. The best makespan values were achieved when using $\rho = 0.7$.

6.4.1.3 Parameter tuning for rGA+VNS and rGA(VNS)

Four parameters were tested for the hybrid rGA+VNS and rGA(VNS) algorithms which included population size, crossover type, mutation type, crossover probability and mutation probability.

The candidate values for the population size were 10, 20 and 30 individuals. The best results were indicated when using a population size of 20 solutions. The experiments showed a very slow improvement rate and that a greater computational time was required to find a good mapping of jobs to resources when increasing the number of individuals from 20 to 30, suggesting that using a large population size is not beneficial for the rGA+VNS and rGA(VNS).

As the permutation-based representation is used, three different crossover operators were applied, which were Order Crossover (OX), Partially Matched Crossover (PMX)

Table 6.1 Neighbourhood structures order testing for rGA+VNS, rACO+VNS, rACO(VNS) and rGA(VNS). The best average makespan results are reported in bold.

Case	Nieghbourhood order	d_u_c_lolo				d_u_i_hihi				d_u_s_lohi			
		rGA+VNS	rACO+VNS	rACO(VNS)	rGA(VNS)	rGA+VNS	rACO+VNS	rACO(VNS)	rGA(VNS)	rGA+VNS	rACO+VNS	rACO(VNS)	rGA(VNS)
1	LMMTM-RMMTM-PTM-PSM	8589.60	8319.80	7922.40	7574.50	25435243.30	24245354.20	23746348.60	23087120.70	958490.10	909977.30	896231.20	887968.50
2	LMMTM-RMMTM-PSM-PTM	8506.30	8057.40	7755.40	7482.90	25111089.90	24128516.20	23472008.20	22805347.10	956923.70	905403.70	893081.90	878337.80
3	LMMTM-PTM-RMMTM-PSM	8566.60	8152.40	7860.30	7562.30	28681255.80	24619500.70	23955627.90	23275245.00	960019.40	918546.90	900358.40	885898.80
4	LMMTM-PTM-PSM-RMMTM	8671.30	8374.50	7950.10	7487.70	25471846.80	24251553.50	23783892.50	23096063.90	959228.80	911751.80	897659.30	881689.50
5	LMMTM-PSM-RMMTM-PTM	8572.10	8201.50	7860.50	7713.70	28589110.80	24607115.30	23939437.90	23269225.30	959269.00	911922.70	897825.70	888280.70
6	LMMTM-PSM-PTM-RMMTM	8502.70	8055.20	7737.70	7481.80	25099250.20	24053421.90	23475989.90	22790272.30	957335.60	905736.40	893503.60	877138.50
7	RMMTM-LMMTM-PTM-PSM	8519.10	8073.30	7811.30	7522.00	25528549.50	24265566.90	23840632.20	23129021.70	961076.80	926154.90	901715.20	884503.70
8	RMMTM-LMMTM-PSM-PTM	8512.70	8058.70	7771.20	7468.00	25129838.50	24131289.10	23483435.50	22916879.30	957174.30	905657.40	893420.10	877815.40
9	RMMTM-PTM-LMMTM-PSM	8564.90	8139.60	7852.70	7702.70	28340815.60	24578572.00	23930632.30	23265335.00	961364.30	927212.40	901801.70	889762.80
10	RMMTM-PTM-PSM-LMMTM	8575.60	8267.40	7903.80	7656.70	27398190.00	24578232.30	23912846.00	23245535.20	959747.20	917926.40	899641.90	880155.60
11	RMMTM-PSM-LMMTM-PTM	8551.90	8113.50	7824.20	7494.50	25558789.00	24301003.80	23849536.80	23160845.60	961002.50	924803.00	901694.40	884092.50
12	RMMTM-PSM-PTM-LMMTM	8488.30	8023.30	7727.30	7460.40	25091512.50	24103898.30	23498944.30	22804519.20	957493.50	906246.60	894212.00	876992.90
13	PTM-LMMTM-RMMTM-PSM	8574.20	8265.30	7891.80	7491.20	26913272.70	24497381.20	23902360.80	23208874.90	960148.10	920459.40	901574.30	887549.40
14	PTM-LMMTM-PSM-RMMTM	8562.10	8120.00	7825.70	7632.20	25857936.10	24420854.10	23873740.10	23192246.50	960597.30	924234.20	901657.90	885332.30
15	PTM-RMMTM-LMMTM-PSM	8572.40	8237.70	7873.70	7570.30	26013095.50	24496150.60	23900073.10	23202236.40	960169.00	921647.40	901586.80	879869.80
16	PTM-RMMTM-PSM-LMMTM	8583.50	8299.60	7907.90	7559.10	25637401.50	24330041.80	23857637.70	23168462.00	959294.00	916399.60	899495.20	887815.40
17	PTM-PSM-LMMTM-RMMTM	8679.60	8394.40	7980.30	7640.60	25232529.00	24135918.00	23626372.00	22995923.40	958332.20	907743.70	895386.00	877588.30
18	PTM-PSM-RMMTM-LMMTM	8672.70	8385.40	7977.30	7585.50	25152887.30	24134552.00	23504434.10	22979726.10	959194.10	911709.50	896973.10	877381.00
19	PSM-LMMTM-RMMTM-PTM	8522.60	8111.90	7817.80	7705.50	25806107.30	24409558.90	23872056.60	23188864.70	961847.10	928063.40	901835.00	877453.60
20	PSM-LMMTM-PTM-RMMTM	8515.20	8060.70	7800.40	7539.60	25649509.30	24378838.70	23865206.90	23182813.20	960434.20	923051.20	901636.70	877951.10
21	PSM-RMMTM-LMMTM-PTM	8642.60	8364.70	7936.70	7497.70	25402437.50	24157937.90	23713808.80	23057770.20	958147.20	907206.60	895316.20	885572.80
22	PSM-RMMTM-PTM-LMMTM	8591.80	8340.60	7926.50	7612.20	25384910.10	24145900.80	23637023.30	23038683.30	957996.80	906257.50	895063.30	877731.30
23	PSM-PTM-LMMTM-RMMTM	**8350.80**	7915.80	7631.20	7447.20	25081661.90	24048143.20	23450419.70	22772568.00	956589.90	**905078.80**	892829.70	876973.40
24	PSM-PTM-RMMTM-LMMTM	8364.20	**7898.50**	**7626.60**	**7440.40**	**25075485.20**	**24042023.70**	**23424252.50**	**22752490.80**	**953553.20**	905079.50	**892737.80**	**876824.80**

Table 6.2 Parameter tuning for rACO+VNS and rACO(VNS) algorithms: α and β. The best average makespan results are reported in bold.

Case	α	β	rACO+VNS			rACO(VNS)		
			d_u_c_lolo	d_u_i_hihi	d_u_s_lohi	d_u_c_lolo	d_u_i_hihi	d_u_s_lohi
1	1	1	7939.10	24222945.30	917053.80	7721.50	23731811.30	901715.20
2	1	5	7949.70	24254306.70	925752.20	7723.70	23776911.90	893420.10
3	1	10	7965.40	24270335.10	926714.40	7735.60	23815160.10	901801.70
4	5	1	7890.20	**24135051.70**	907201.50	7665.50	23482233.40	900358.40
5	5	5	7920.50	24146898.70	910134.10	7712.60	23514521.40	897825.70
6	5	10	7927.70	24146973.60	912630.70	7715.90	23544120.50	893503.60
7	10	1	**7875.60**	24136720.30	**905423.30**	**7642.30**	**23471346.80**	**896231.20**
8	10	5	7906.20	24137527.10	905877.60	7710.50	23495090.20	893081.90
9	10	10	7909.10	24140477.20	909566.20	7711.60	23507823.70	897659.30

and Cycle Crossover (CX) with the best results being achieved when using CX crossover operator, as illustrated in Fig. 6.2, while Table 6.3 reports the values of other parameters used to compare the performance of different crossover operators.

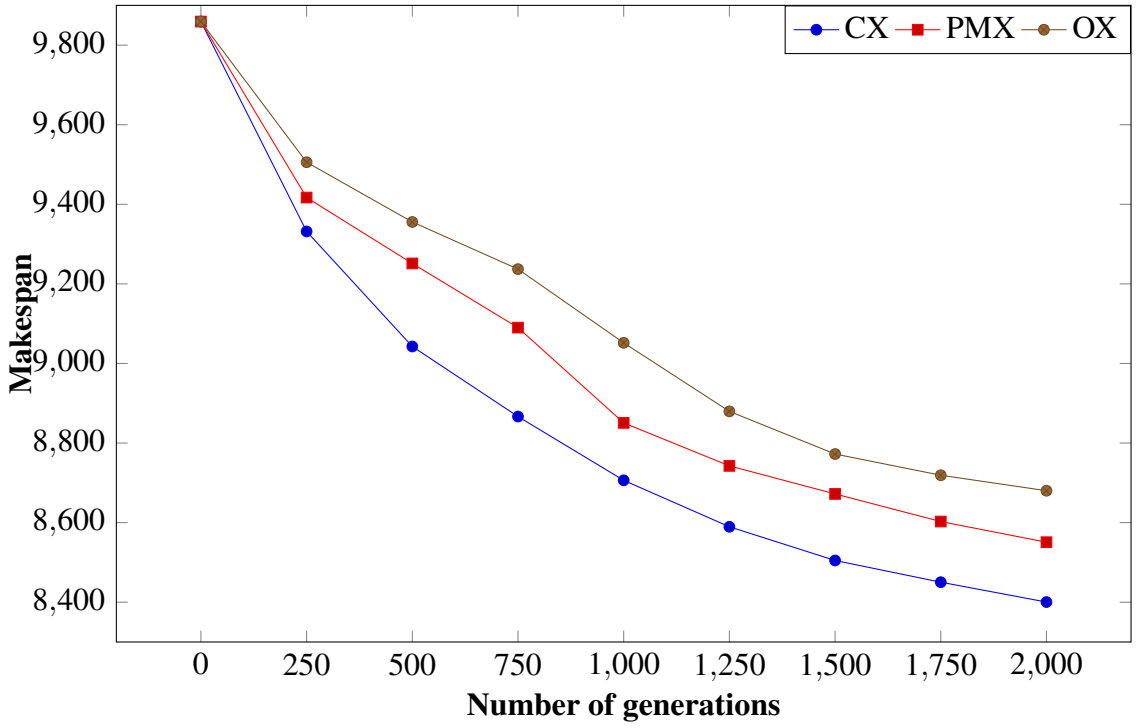

Fig. 6.2 Parameter tuning for different crossover operators of rGA(VNS) using d-u-c-lolo instance.

Table 6.3 Parameter values used for comparing the performance of different crossover operators.

Seeding method	min-min algorithm
Number of generations	2000
Probability of crossover	0.7
Population size	20
Selection operator	N-Tournament, N = 4
Mutation operator	VNS
Probability of mutation	0.8
Replacement operator	Steady-State

In similar fashion, four different mutation operators were used, which were Random move, Random swap, Best move and Best swap with the best results being achieved when using the Best swap operator, as illustrated in Fig. 6.3, while Table 6.4 reports the values of other parameters used to compare the performance of different mutation operators.

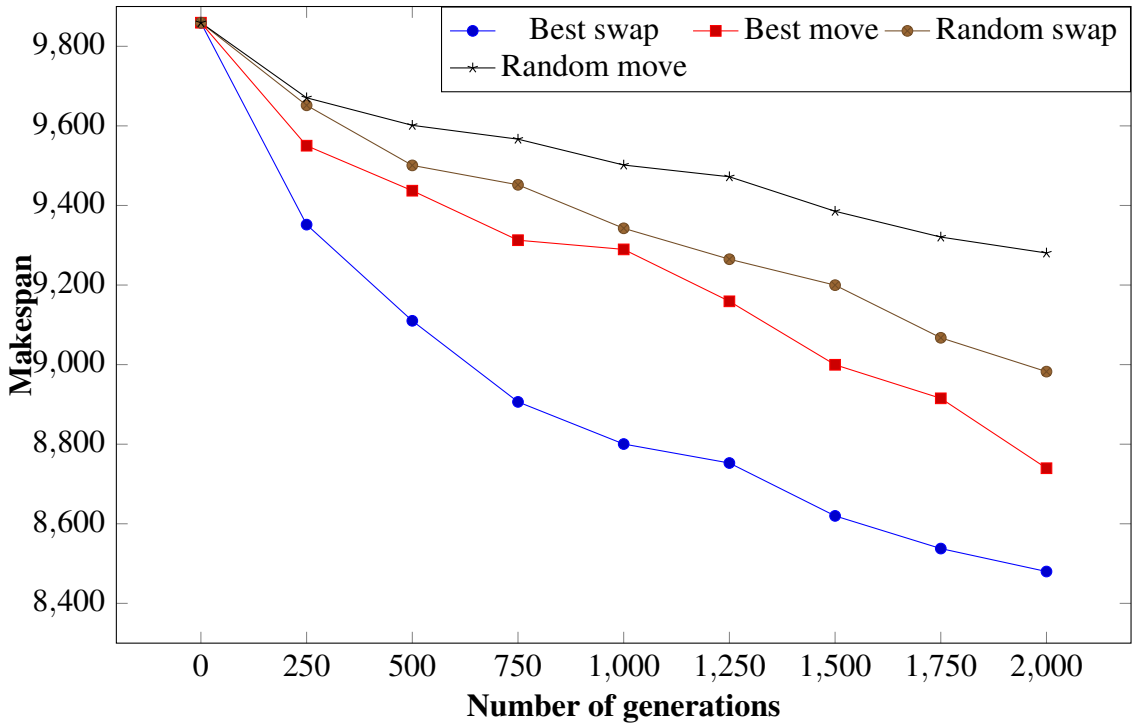

Fig. 6.3 Parameter tuning for different mutation operators of rGA+VNS using d-u-c-lolo instance.

Table 6.4 Parameter values used for comparing the performance of different mutation operators.

Seeding method	min-min algorithm
Number of generations	2000
Probability of crossover	0.7
Population size	20
Selection operator	N-Tournament, N = 4
Crossover operator	CX
Probability of mutation	0.8
Replacement operator	Steady-State

Finally, the probability of crossover and mutation were examined. A considerable number of studies in the literature suggested high crossover and mutation probabilities [25] [170] [169] [163]; therefore, the candidate values used were 0.7, 0.8 and 0.9. The best result was recorded when using pc = 0.7 and pm = 0.8, as shown in Fig. 6.4 and Fig. 6.5.

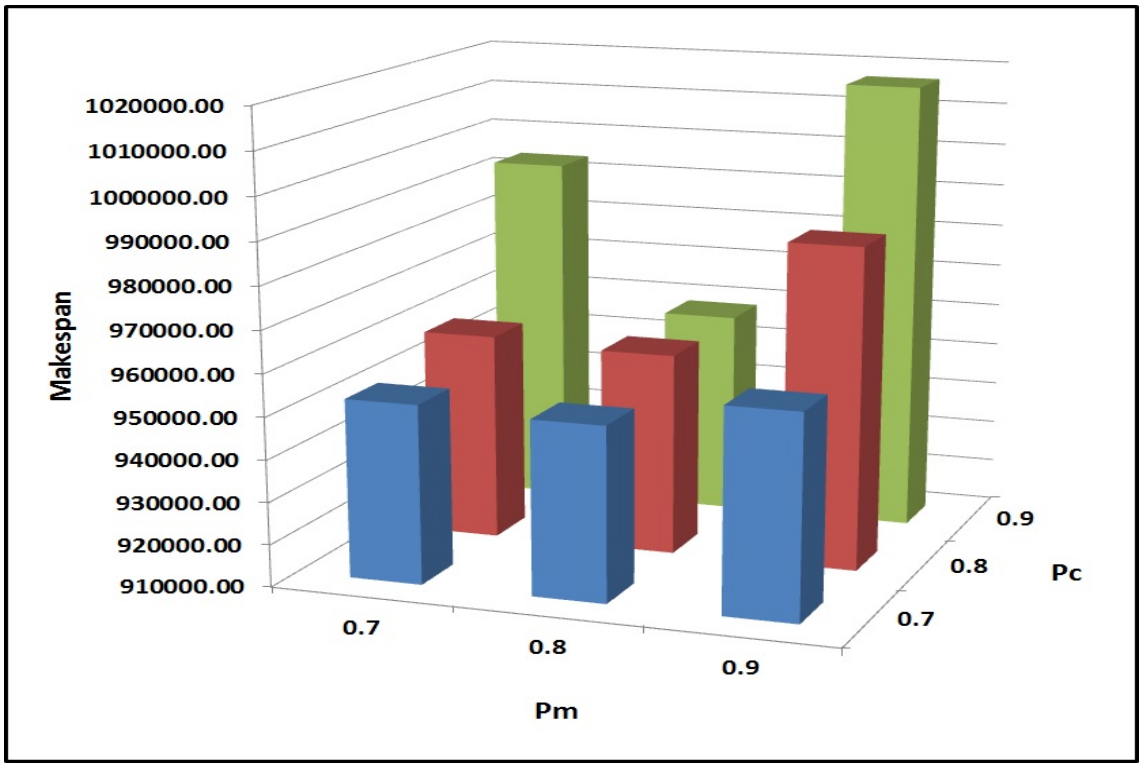

Fig. 6.4 Analysis of rGA+VNS operators probabilities using d-u-s-lohi instance.

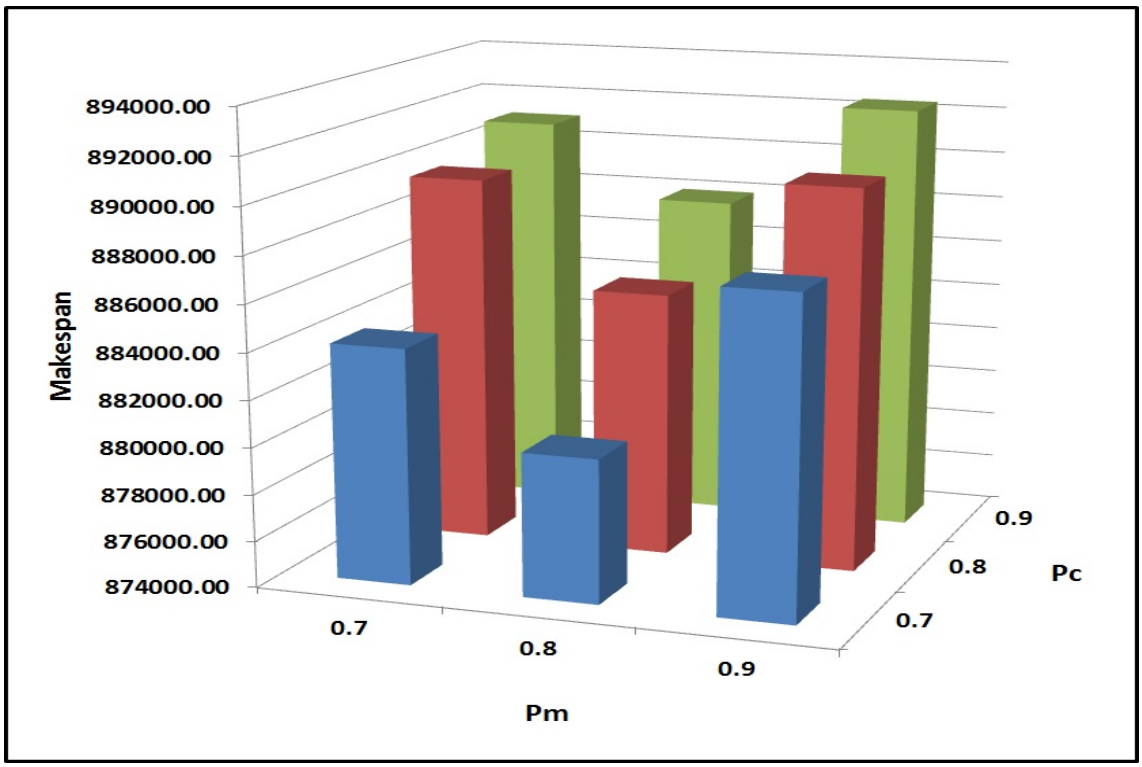

Fig. 6.5 Analysis of rGA(VNS) operators probabilities using d-u-s-lohi instance.

6.4.2 Experimental results

To study the effects of applying rescheduling, each algorithm has two versions, namely without rescheduling and with rescheduling. The algorithms without rescheduling schedule each group separately, while the algorithms with rescheduling follow the scenario described in Algorithm 6.1. For both versions, all algorithms were allowed to run using the stopping times listed in Table 6.5, which are the same to the ones used for the static dataset of Braun *et al.* To obtain the best, average and standard deviation values, each algorithm was executed 10 times for each problem instance. Java language was used to implement the proposed methods. An Intel i5-4570 CPU @ 3.20 GHz pc with 8 GB RAM has been used to carry out all the experiments reported in this chapter.

Table 6.5 Stopping times for the proposed methods for 512x16 dynamic dataset instances.

Algorithm	stopping time	total time
rGA(VNS)	90 seconds	90 seconds
rGA+VNS	80 seconds for GA 10 seconds for VNS	90 seconds
rACO(VNS)	9 minutes	9 minutes
rACO+VNS	8 minutes and 50 seconds for ACO 10 seconds for VNS	9 minutes

Table 6.6 provides the results of applying the various proposed hybrid methods compared to each other and the min-min algorithm [77]. The best results are indicated in bold, which show clearly that rGA(VNS) outperforms all other approaches in all instances. The rGA(VNS) algorithm is expected to find high-quality schedules in any single execution since it has very small standard deviation values in the range [0.11, 0.16]. The table also shows that rACO(VNS) which showed the second-best performance after rGA(VNS) with relatively small standard deviation values between 0.15 and 0.24. Both rGA(VNS) and rACO(VNS) are strongly coupled hybrid meta-heuristic rescheduling methods. In general, the relative performance order of the implemented methods from best to worst was: (1)rGA(VNS), (2)rACO(VNS), (3)GA(VNS), (4)rACO+VNS, (5)ACO(VNS), (6)ACO+VNS, (7)rGA+VNS, (8)GA+VNS, (9)rmin-min, and (10)min-min.

Table 6.7 shows the improvement percentages of each rescheduling algorithm over their corresponding traditional methods (without rescheduling). It is clear that the use of rescheduling improves the makespan results for all algorithms with a minimum average improvement of 2.13% and 2.17% (for the best and average cases respectively) and a maximum average of 4.93% and 5.01% (for the best and average cases respectively). The two sample t-test with unequal variants was performed to statistically analyse the performance of the proposed hybrid methods. Table 6.7 reports the corresponding p-value for each problem instance and each rescheduling algorithm which were less than 0.05, and hence we can consider the improvement of the rescheduling-based algorithms over non-rescheduling-based algorithms in terms of makespan to be statistically significant.

Table 6.8 shows the improvement percentages of rGA(VNS) over the other methods. rGA(VNS) shows a better improvement percentage over all the methods compared with a minimum average improvement of 2.06% (for the best and average cases) and a maximum average improvement of 34.98% and 20.59% (for the best and average cases respectively). These results indicate that GA(VNS) represents not only the new state-of-the-art sequential hybrid algorithms for the static job scheduling problem in grid computing, its corresponding dynamic method, rGA(VNS), is significantly better than all other methods in the dynamic version of the problem as well.

6.5 Further discussion

This chapter discussed the application of hybrid meta-heuristic rescheduling algorithms for the dynamic version of the job scheduling problem in grid computing. The achieved results show that the use of rescheduling strategy by the various schedulers provides statistically significant improvements in terms of makespan compared to their corresponding traditional (without rescheduling) methods. The reported results also show that the two strongly coupled hybrid meta-heuristic rescheduling methods, rGA(VNS) and rACO(VNS), performed

Table 6.6 Makespan results for the 512x16 dynamic dataset instances.

		Instance											
Algorithm		d_u_c_hihi	d_u_c_hilo	d_u_c_lohi	d_u_c_lolo	d_u_i_hihi	d_u_i_hilo	d_u_i_lohi	d_u_i_lolo	d_u_s_hihi	d_u_s_hilo	d_u_s_lohi	d_u_s_lolo
min-min	Res	21165908.30	289094.00	570180.20	9859.30	30863194.40	330691.90	1210593.50	12367.20	65817894.80	553740.60	2407582.50	14566.30
rmin-min	Res	20809672.30	285207.10	569659.90	9201.40	30673618.30	323130.50	1118597.70	12342.60	58447697.60	497837.60	2234119.00	14336.10
	best	17103042.00	268872.30	495374.90	8502.70	25402437.50	268192.00	852255.60	9422.50	30886119.80	307417.50	950597.30	10476.40
GA+VNS	avg	17159899.31	270534.91	498342.628	8541.2	25536091.58	269332.22	856910.68	9470.98	31120140.06	309228.76	955375.13	10537.36
	σ	0.28%	0.28%	0.28%	0.29%	0.26%	0.24%	0.25%	0.29%	0.27%	0.29%	0.28%	0.27%
	best	16556469.10	257747.30	481056.40	8267.40	24885173.40	266229.60	830749.00	9324.40	30100694.00	302954.60	946482.00	10281.20
rGA+VNS	avg	16603952.86	259091.88	483000.244	8335.12	25068333.8	267365.98	835364.18	9366.15	30271678.51	303899.01	951163.77	10324.95
	σ	0.26%	0.25%	0.27%	0.28%	0.26%	0.24%	0.24%	0.21%	0.26%	0.27%	0.26%	0.21%
	best	16401492.90	251921.80	471410.70	8060.70	24496150.60	263113.50	822317.20	9181.60	29618156.60	298809.80	924234.20	10198.10
ACO+VNS	avg	16441782.86	253127.19	473340.88	8115.59	24618879.15	264148.77	826200.66	9230.17	29770039.79	299861.61	928516.61	10233.78
	σ	0.18%	0.27%	0.28%	0.29%	0.27%	0.27%	0.28%	0.29%	0.26%	0.26%	0.26%	0.20%
	best	15806938.80	231506.80	464843.80	7825.70	23939437.90	258919.20	802276.50	9080.30	28916245.80	290213.50	902516.30	9968.50
rACO+VNS	avg	15860226.33	232971.56	466102.702	7867.53	24033282.82	259996.87	805521.37	9103.25	29056693.81	292119.09	904622.65	9998.01
	σ	0.18%	0.26%	0.20%	0.28%	0.22%	0.19%	0.23%	0.18%	0.19%	0.26%	0.19%	0.18%
	best	16037906.80	236859.40	467417.97	7936.70	24145900.80	260739.80	813374.80	9122.30	29236827.50	294663.40	909977.30	10052.00
ACO(VNS)	avg	16116283.61	238463.13	468640.627	7982.53	24270661.06	262104.78	816349.18	9143.98	29412680.37	296400.62	913139.22	10086.79
	σ	0.19%	0.19%	0.16%	0.21%	0.24%	0.23%	0.25%	0.18%	0.25%	0.24%	0.25%	0.22%
	best	15417575.00	224797.40	457715.20	7570.30	23275245.00	255927.90	788844.70	8955.90	28252220.30	286530.40	889911.90	9789.30
rACO(VNS)	avg	15501399.39	225676.57	458651.852	7603.39	23408812.45	256775.5	792767.48	8975.86	28391396.37	287185.39	892686.25	9823.3
	σ	0.18%	0.19%	0.15%	0.21%	0.24%	0.21%	0.18%	0.19%	0.22%	0.17%	0.18%	0.17%
	best	15623467.80	226882.60	460406.40	7716.60	23713808.80	257709.80	796685.40	9014.60	28696865.60	288077.20	896231.20	9858.30
GA(VNS)	avg	15673599.78	227714.97	461804.591	7737.31	23810096.57	258409.52	798761.84	9040.3	28791592.71	288977.29	899224.33	9885.84
	σ	0.16%	0.16%	0.16%	0.18%	0.17%	0.14%	0.16%	0.18%	0.18%	0.18%	0.18%	0.18%
	best	**15065687.50**	**220828.90**	**445649.70**	**7421.10**	**22608940.20**	**248884.00**	**772937.50**	**8825.90**	**27713994.50**	**282258.00**	**870926.50**	**9647.90**
rGA(VNS)	avg	**15146746.7**	**221151.54**	**447417.531**	**7437.37**	**22723534.97**	**250462.88**	**775911.75**	**8846.85**	**27810189.35**	**283013.72**	**876641.73**	**9671.64**
	σ	0.14%	0.11%	0.15%	0.16%	0.16%	0.14%	0.16%	0.14%	0.15%	0.16%	0.16%	0.15%

Table 6.7 Best and average improvement percentages, and statistical analysis of rescheduling methods over their corresponding non-rescheduling methods for the 512x16 dynamic dataset instances.

		Instance												
Algorithm		d_u_c_hihi	d_u_c_hilo	d_u_c_lohi	d_u_c_lolo	d_u_i_hihi	d_u_i_hilo	d_u_i_lohi	d_u_i_lolo	d_u_s_hihi	d_u_s_hilo	d_u_s_lohi	d_u_s_lolo	Avg
rmin-min	Res	1.68	1.34	0.09	6.67	0.61	2.29	7.60	0.20	11.20	10.10	7.20	1.58	4.21
	best	3.20	4.14	2.89	2.77	2.04	0.73	2.52	1.04	2.54	1.45	0.43	1.86	2.13
rGA+VNS	avg	3.24	4.23	3.08	2.41	1.83	0.73	2.51	1.11	2.73	1.72	0.44	2.02	2.17
	p-value	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	0.00866	$<10^{-5}$	
	best	3.62	8.10	1.39	2.92	2.27	1.59	2.44	1.10	2.37	2.88	2.35	2.25	2.77
rACO+VNS	avg	3.54	7.96	1.53	3.06	2.38	1.57	2.50	1.38	2.40	2.58	2.57	2.30	2.81
	p-value	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	
	best	3.87	5.09	2.08	4.62	3.61	1.85	3.02	1.82	3.37	2.76	2.21	2.61	3.07
rACO(VNS)	avg	3.82	5.36	2.13	4.75	3.55	2.03	2.89	1.84	3.47	3.11	2.24	2.61	3.15
	p-value	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	
	best	3.57	2.67	3.21	3.83	4.66	3.42	2.98	2.09	3.43	2.02	2.82	2.13	3.07
rGA(VNS)	avg	3.36	2.88	3.12	3.88	4.56	3.08	2.86	2.14	3.41	2.06	2.51	2.17	3.00
	p-value	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	$<10^{-5}$	

Table 6.8 Best and average improvement percentages of the rGA(VNS) algorithm over other methods for the 512x16 dynamic dataset instances.

		Instance												
Algorithm		d_u_c_hihi	d_u_c_hilo	d_u_c_lohi	d_u_c_lolo	d_u_i_hihi	d_u_i_hilo	d_u_i_lohi	d_u_i_lolo	d_u_s_hihi	d_u_s_hilo	d_u_s_lohi	d_u_s_lolo	Avg
min-min	Res	28.82	23.61	21.84	24.73	26.74	24.74	36.15	28.63	57.89	49.03	63.83	33.77	34.98
rmin-min	Res	27.60	22.57	21.77	19.35	26.29	22.98	30.90	28.49	52.58	43.30	61.02	32.70	32.46
GA+VNS	best	11.91	17.87	10.04	12.72	11.00	7.20	9.31	6.33	10.27	8.18	8.38	7.91	10.09
	avg	11.73	18.25	10.22	12.92	11.01	7.01	9.45	6.59	10.64	8.48	8.24	8.22	10.25
rGA+VNS	best	9.00	14.32	7.36	10.24	9.15	6.52	6.96	5.35	7.93	6.83	7.98	6.16	8.15
	avg	8.78	14.64	7.37	10.77	9.35	6.32	7.12	5.54	8.13	6.87	7.83	6.33	8.35
ACO+VNS	best	8.14	12.34	5.46	7.93	7.70	5.41	6.00	3.87	6.43	5.54	5.77	5.40	6.67
	avg	7.88	12.63	5.48	8.36	7.70	5.18	6.09	4.15	6.58	5.62	5.59	5.49	6.73
rACO+VNS	best	4.69	4.61	4.13	5.17	5.56	3.88	3.66	2.80	4.16	2.74	3.50	3.22	4.01
	avg	4.50	5.07	4.01	5.47	5.45	3.67	3.68	2.82	4.29	3.12	3.09	3.26	4.04
ACO(VNS)	best	6.06	6.77	4.66	6.50	6.37	4.55	4.97	3.25	5.21	4.21	4.29	4.02	5.07
	avg	6.02	7.26	4.53	6.83	6.37	4.44	4.95	3.25	5.45	4.52	4.00	4.12	5.14
rACO(VNS)	best	2.28	1.77	2.64	1.97	2.86	2.75	2.02	1.45	1.91	1.49	2.13	1.44	2.06
	avg	2.29	2.01	2.45	2.18	2.93	2.46	2.13	1.44	2.05	1.45	1.80	1.54	2.06
GA(VNS)	best	3.57	2.67	3.21	3.83	4.66	3.42	2.98	2.09	3.43	2.02	2.82	2.13	3.07
	avg	3.36	2.88	3.12	3.88	4.56	3.08	2.86	2.14	3.41	2.06	2.51	2.17	3.00

better than their corresponding loosely coupled schedulers. It is worth noting that the version of the dynamic job scheduling problem discussed in this chapter considered the change in the number of jobs at runtime only, while the number of resources is the same. For further work, it will be interesting to examine the change in the number of jobs and resources and the work presented in this chapter can serve as reference in this direction.

6.6 Summary

In this chapter, the application of various hybrid meta-heuristic rescheduling algorithms for solving the dynamic job scheduling problem in grid computing in terms of minimising the makespan was discussed. The dynamic version of the job scheduling problem in grid computing that has been considered here involves the scenario in which blocks of independent jobs arrive to the grid system at different arrival times. The rescheduling strategy, which involves several calls of the job scheduler at various intervals of time, is employed to solve this dynamic problem. The dynamic job scheduling problem formulation, the dynamic simulation model and how to apply rescheduling were introduced. Moreover, a special benchmark has been created for the dynamic job scheduling problem to validate the performance of the proposed hybrid meta-heuristic rescheduling algorithms. The experimental results showed that the algorithm which apply rescheduling provides better makespan results than its corresponding non-rescheduling algorithm. Furthermore, the two strongly coupled meta-heuristic rescheduling algorithms, namely rGA(VNS) and rACO(VNS), provided promising and significant improvements in terms of minimising the makespan, which are outperforming their corresponding loosely coupled rescheduling and traditional methods. These improvements suggest that the two strongly coupled meta-heuristics with aid of rescheduling operations have the ability to efficiently tackle the dynamic job scheduling problem in grid computing.

Chapter 7

Conclusions and Future Work

This chapter summarizes the results of this work and draws some conclusions. Additionally, the main contributions of the research work presented in this thesis are also listed. Finally, areas for further research are identified.

7.1 Conclusions

The mapping of jobs to resources or job scheduling in distributed and heterogeneous environments such as grid computing systems is considered one of the most significant and difficult tasks. The overall performance of such systems can be improved significantly by using an effective job scheduler. The job scheduling in grid computing shares the property of being an NP-hard problem with conventional distributed systems. However, in the former systems, it is particularly complex as it is dynamic, multi-objective and has a high degree of heterogeneity in terms of jobs and resources. Therefore, and to cope in practice with its difficulty and complexity, the use of meta-heuristics is necessary. ACO and GA are robust search methods which have been used to successfully solve this problem. However, the results achieved by these methods could be further improved by combining them with other meta-heuristic approaches.

In this thesis, two meta-heuristic methods, ACO and GA, have been hybridized with a novel VNS in loosely and strongly coupled fashions to tackle the static and dynamic independent batch job scheduling problems in grid computing. The new high-level algorithms inherit the best characteristics of the combined methods. Four new neighbourhood structures and a modified PALS have been proposed for the novel VNS, which use the concepts of move and transfer of some jobs to or from the problem resource, which is the resource that has a local makespan equal to the total makespan of the solution. Through the use of these structures and the modified local search, VNS improves the performance of the ACO and GA algorithms by introducing diversity to the colony and the population, respectively, and by exploring new parts of the state space of the problem.

To evaluate the performance of the proposed hybrid methods to the static version of the problem, the ETC model has been used. Three different well-known datasets have been used to perform several experiments. The experimental results show that the strongly coupled hybrid meta-heuristic, GA(VNS), achieved results that were significantly better than other selected approaches from the literature for all three benchmarks used in terms of minimising the makespan; therefore, we can claim that it represents the new state-of-the-art sequential hybrid algorithm for job scheduling in grid computing. With very low standard deviation values, it should be expected that GA(VNS) can find high-quality schedules in any single run. Moreover, GA(VNS) achieved results that show the smallest gap with the lower bound in all the problem instances examined in this study. On the other hand, the other strongly coupled hybrid meta-heuristic, ACO(VNS), was almost the second-best algorithm in terms of makespan results; however, it needed a longer time to construct high-quality solutions. For relatively small problem instances, the result for ACO(VNS) were very close to the ones achieved by GA(VNS); however, as the dataset size increases, the quality of the solutions found by ACO(VNS) decreases, which means that longer times will be needed to improve the results. The thesis also examined the loosely coupled hybridisation of ACO and GA with VNS. The first loosely coupled hybrid meta-heuristic, ACO+VNS, was almost the third-best algorithm in terms of makespan results; however, similar to ACO(VNS), it needed a longer time to construct high-quality solutions. The second loosely coupled hybrid meta-heuristic, GA+VNS, was almost the fourth-best algorithm in terms of makespan results. The main feature of loosely coupled hybrid methods is that the identity of the combined algorithms is preserved. This means that no overlapping between the combined algorithms is available, which leads to the situation that if one algorithm sticks in a local minimum, the following algorithm will try to escape from it and improve the solution. However, this improvement is limited mainly due to the fact that the following algorithm may also be trapped in local minima. This situation is not exist in the strongly coupled algorithms as if the main algorithm sticks in a local minimum, it will escape from it when it calls the supporting algorithm which passes the control back to main algorithm again and so on.

Moreover, the application of various hybrid meta-heuristic rescheduling algorithms for solving the dynamic job scheduling problem in grid computing in terms of minimising the makespan was also discussed in this thesis. The dynamic version of the job scheduling problem in grid computing that has been considered in this research work involves the scenario in which blocks of independent jobs arrive to the grid system at different arrival times. The rescheduling strategy, which involves several calls of the job scheduler at various intervals of time, is employed to solve this dynamic problem. The dynamic job scheduling problem formulation, the dynamic simulation model and how to apply rescheduling were introduced. Moreover, a special benchmark has been created for the dynamic job scheduling problem to validate the performance of the proposed hybrid meta-heuristic

rescheduling algorithms. The experimental results showed that the algorithm which apply rescheduling provides better makespan results than its corresponding none rescheduling algorithm. Furthermore, the two strongly coupled meta-heuristic rescheduling algorithms, namely rGA(VNS) and rACO(VNS), provided promising and significant improvements in terms of minimising the makespan, which are outperforming their corresponding loosely coupled rescheduling and traditional methods. These improvements suggest that the two strongly coupled meta-heuristics with aid of rescheduling operations have the ability to efficiently tackle the dynamic job scheduling in grid computing.

From the research that has been conducted, it is possible to summarise the following main contributions:

1. An analyse of the use of various traditional, heuristic, meta-heuristic and hybrid meta-heuristics approaches for solving the job scheduling problem in grid computing, was presented.

2. A new VNS meta-heuristic for the job scheduling problem in grid computing, which uses some effective and carefully designed neighbourhood structures and a powerful local search to explore different regions on the state space of the problem, was developed.

3. Several novel hybrid meta-heuristic schedulers, in loosely and strongly coupled fashions, that uses the newly proposed VNS to produce high quality schedules in a reasonable time, were designed and implemented.

4. New state-of-the-art sequential hybrid algorithms for job scheduling in grid computing that could serve as a reference in the field, were provided.

5. New problem instances, that follow a well-known methodology, to model dynamic job scheduling in grid computing, were generated.

6. A dynamic scenario which uses the rescheduling technique to simulate the dynamic job scheduling problem, was introduced.

7. The introduced dynamic scenario was used to evaluate the performance of the proposed hybrid schedulers.

7.2 Future work

Although the proposed methods seem promising approaches to scheduling in grid computing systems, the work presented in this line of research can be extended in various directions.

In the current GA-based schedulers, the focus was on improving the mutation operator in which the use of VNS showed a great impact. We are in the process of developing an adaptive crossover operator for the proposed GA-based methods. It is expected that the new crossover operator will significantly improve the performance of the proposed GA-based schedulers, especially the loosely coupled one.

Furthermore, research into solving the job scheduling problem in grid computing using other meta-heuristic approaches, such as PSO, both as a stand-alone and as a hybrid algorithm is already underway. The investigation of other meta-heuristics will allow us to more generalise the hybridisation concepts and impacts, and to a great extent, to construct a general framework to design and develop hybrid algorithms to solve other optimisation problems.

The suggested methods in this work addressed the minimisation of a single objective, which is the makespan. Adding another objective, such as flowtime and cost, will convert the problem into a multi-objective one. Therefore, future work needs to be carried out to establish whether the proposed methods can tackle the multi-objective job scheduling problem in grid computing or not. Moreover, the makespan, cost and flowtime objectives are related to grid computing system performance and users' QoS requirements, it will be interesting to study the behaviour of the proposed hybrid meta-heuristic algorithms in terms of energy- and security- based and/or performance- and QoS- based objectives.

On the other hand, the proposed hybrid methods in this thesis tackled the independent job scheduling, which assumes no relations between the submitted jobs. Extend the hybrid schedulers suggested in this work to include the dependent version of the job scheduling problem is a promising line of research to develop the current. Furthermore, multi-objective dependent job scheduling is also another direction which considers even more complex version of the scheduling problem.

Moreover, the hybrid methods proposed in this study are sequential; it would be interesting to examine their performances in the parallel mode. This more likely leads to many changes and modifications in the algorithm side of the proposed hybrid algorithms to cope with the parallel requirements.

Although the achieved results of applying the proposed hybrid schedulers to the dynamic version of the problem were promising, more research efforts are required. Recalling that the version of the dynamic job scheduling problem discussed in this thesis considered the change in the number of jobs at runtime only, while the number of resources is the same. For further work, it will be interesting to examine the change in the number of jobs and resources to which the work presented in this thesis can serve as reference in this direction. In addition, other types of dynamic scheduling problems in grid computing can also be investigated such as dynamic dependent job scheduling, dynamic multi-objective independent job scheduling and dynamic multi-objective dependent job scheduling problems.

More broadly, the prospect of being able to apply the proposed methods in this work to other distributed and heterogeneous environments, such as cloud computing systems, serves as a continuous incentive for future research.

Bibliography

[1] Ajith Abraham, Rajkumar Buyya, and Baikunth Nath. Nature's heuristics for scheduling jobs on computational grids. In *The 8th IEEE international conference on advanced computing and communications (ADCOM 2000)*, pages 45–52, 2000.

[2] Ajith Abraham, He Guo, and Hongbo Liu. Swarm intelligence: foundations, perspectives and applications. In *Swarm Intelligent Systems*, pages 3–25. Springer, 2006.

[3] Ajith Abraham, Hongbo Liu, and Mingyan Zhao. Particle swarm scheduling for work-flow applications in distributed computing environments. In *Metaheuristics for Scheduling in Industrial and Manufacturing Applications*, pages 327–342. Springer, 2008.

[4] Lucio Agostinho, Guilherme Feliciano, Leonardo Olivi, Eleri Cardozo, and Eliane Guimaraes. A bio-inspired approach to provisioning of virtual resources in federated clouds. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 598–604. IEEE, 2011.

[5] Imtiaz Ahmad and Muhammad K Dhodhi. Multiprocessor scheduling in a genetic paradigm. *Parallel Computing*, 22(3):395–406, 1996.

[6] Enrique Alba and Gabriel Luque. A new local search algorithm for the dna fragment assembly problem. *Evolutionary Computation in Combinatorial Optimization*, pages 1–12, 2007.

[7] Abdelkamel Ben Ali, Gabriel Luque, Enrique Alba, and Kamal E Melkemi. An improved problem aware local search algorithm for the dna fragment assembly problem. *Soft Computing*, 21(7):1709–1720, 2017.

[8] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, and Debra Hensgen. Task execution time modeling for heterogeneous computing systems. In *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pages 185–199. IEEE, 2000.

[9] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Representing task and machine heterogeneities for heterogeneous computing systems. *Tamkang Journal of Science and Engineering*, 3(3):195–207, 2000.

[10] Mustafa Muwafak Alobaedy and Ku Ruhana Ku-Mahamud. Scheduling jobs in computational grid using hybrid acs and ga approach. In *Computing, Communications and IT Applications Conference (ComComAp), 2014 IEEE*, pages 223–228. IEEE, 2014.

[11] Soheil Anousha, Shoeib Anousha, and Mahmood Ahmadi. A new heuristic algorithm for improving total completion time in grid computing. In *Multimedia and Ubiquitous Engineering*, pages 17–26. Springer, 2014.

[12] P Deepan Babu and T Amudha. A novel genetic algorithm for effective job scheduling in grid environment. In *Computational Intelligence, Cyber Security and Computational Models*, pages 385–393. Springer, 2014.

[13] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Software: Practice and Experience*, 32 (15):1437–1466, 2002.

[14] Amid Khatibi Bardsiri and Seyyed Mohsen Hashemi. A comparative study on seven static mapping heuristics for grid scheduling problem. *International Journal of Software Engineering and Its Applications*, 6(4):247–256, 2012.

[15] Fran Berman, Geoffrey Fox, Tony Hey, and Anthony JG Hey. *Grid computing: making the global infrastructure a reality*, volume 2. John Wiley and sons, 2003.

[16] Christian Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

[17] Christian Blum, Jakob Puchinger, Günther R Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[18] Ilhem BoussaïD, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[19] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, Debra Hensgen, et al. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[20] Tracy D Braun, Howard Jay Siegel, and Anthony A Maciejewski. Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 8–pp. IEEE, 2001.

[21] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. An improved ant system algorithm for thevehicle routing problem. *Annals of operations research*, 89: 319–328, 1999.

[22] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[23] Rajkumar Buyya and Srikumar Venugopal. A gentle introduction to grid computing and technologies. *CSI Communications*, 2:R3, 2005.

[24] Javier Carretero and Fatos Xhafa. Use of genetic algorithms for scheduling jobs in large scale grid applications. *Technological and Economic development of Economy*, 12(1):11–17, 2006.

[25] Javier Carretero, Fatos Xhafa, and Ajith Abraham. Genetic algorithm based schedulers for grid computing systems. *International Journal of Innovative Computing, Information and Control*, 3(6):1–19, 2007.

[26] Anand K Chaturvedi and Rajendra Sahu. New heuristic for scheduling of independent tasks in computational grid. *International Journal of Grid and Distributed Computing*, 4(3):25–36, 2011.

[27] S Chick, PJ Sánchez, D Ferrin, and DJ Morrice. Fast simulation model for grid scheduling using hypersim. In *WSC*, volume 1, page 1494. WSG/SIGSIM, 1998.

[28] P Cyril Daisy Christina and D Doreen Hephzibah Miriam. Adaptive task scheduling based on multi criterion ant colony optimization in computational grids. In *Recent Trends In Information Technology (ICRTIT), 2012 International Conference on*, pages 185–190. IEEE, 2012.

[29] Chao-Hsien Chu, G Premkumar, and Hsinghua Chou. Digital data networks design using genetic algorithms. *European Journal of Operational Research*, 127(1): 140–158, 2000.

[30] Irina Ciornei and Elias Kyriakides. Hybrid ant colony-genetic algorithm (gaapi) for global continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1):234–245, 2012.

[31] Tatjana Davidovic, Pierre Hansen, and Nenad Mladenovic. Variable neighborhood search for multiprocessor scheduling problem with communication delays. In *Proc. MIC*, volume 4, pages 737–741, 2001.

[32] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *IJCAI*, volume 85, pages 162–164, 1985.

[33] Kenneth Alan De Jong. *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[34] J-L Deneubourg, Serge Aron, Simon Goss, and Jacques M Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2): 159–168, 1990.

[35] Marco Dorigo. *Optimization, learning and natural algorithms*. PhD thesis, Politecnico di Milano, 1992.

[36] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[37] Marco Dorigo and Thomas Stützle. The ant colony optimization metaheuristic: Algorithms, applications, and advances. In *Handbook of metaheuristics*, pages 250–285. Springer, 2003.

[38] Marco Dorigo, Mauro Birattari, et al. Swarm intelligence. *Scholarpedia*, 2(9):1462, 2007.

[39] Werner Dubitzky and Francisco Azuaje. A genetic algorithm and growing cell structure approach to learning case retrieval structures. In *Soft computing in case based reasoning*, pages 115–146. Springer, 2001.

[40] Jayne Eaton. *Ant Colony Optimisation for Dynamic and Dynamic Multi-objective Railway Rescheduling Problems*. PhD thesis, School of Computer Science and Informatics, De Montfort University, 2017.

[41] Jayne Eaton and Shengxiang Yang. Dynamic railway junction rescheduling using population based ant colony optimisation. In *Computational Intelligence (UKCI), 2014 14th UK Workshop on*, pages 1–8. IEEE, 2014.

[42] Jayne Eaton, Shengxiang Yang, and Michalis Mavrovouniotis. Ant colony optimization with immigrants schemes for the dynamic railway junction rescheduling problem with multiple delays. *Soft Computing*, 20(8):2951–2966, 2016.

[43] Jayne Eaton, Shengxiang Yang, and Mario Gongora. Ant colony optimization for simulated dynamic multi-objective railway junction rescheduling. *IEEE Transactions on Intelligent Transportation Systems*, 18(11):2980–2992, 2017.

[44] El-Sayed M El-Alfy and Wasan Shaker Awad. Computational intelligence paradigms: An overview. In *Improving Information Security Practices through Computational Intelligence*, pages 1–27. IGI Global, 2016.

[45] Andries P Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley & Sons, Inc., 2007.

[46] Larry J Eshelman. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In *Foundations of genetic algorithms*, volume 1, pages 265–283. Elsevier, 1991.

[47] Kobra Etminani and M Naghibzadeh. A min-min max-min selective algorihtm for grid task scheduling. In *Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on*, pages 1–7. IEEE, 2007.

[48] Kobra Etminani, Mahmaud Naghibzadeh, and Noorali Raeeji Yanehsari. A hybrid min-min max-min algorithm with improved performance. *Department of Computer Engineering, University of Mashad*, 2009.

[49] Conor Fahy, Shengxiang Yang, and Mario Gongora. Finding multi-density clusters in non-stationary data streams using an ant colony with adaptive parameters. In *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pages 673–680. IEEE, 2017.

[50] Conor Fahy, Shengxiang Yang, and Mario Gongora. Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams. *IEEE Transactions on Cybernetics*, 2018.

[51] Geoffrey Falzon and Maozhen Li. Evaluating heuristics for grid workflow scheduling. In *Natural Computation, 2009. ICNC'09. Fifth International Conference on*, volume 4, pages 227–231. IEEE, 2009.

[52] Geoffrey Falzon and Maozhen Li. Enhancing genetic algorithms for dependent job scheduling in grid computing environments. *The Journal of Supercomputing*, 62(1): 290–314, 2012.

[53] Geoffrey Falzon and Maozhen Li. Enhancing list scheduling heuristics for dependent job scheduling in grid computing environments. *The Journal of Supercomputing*, 59(1):104–130, 2012.

[54] Stefka Fidanova and Mariya Durchova. Ant algorithm for grid scheduling problem. In *International Conference on Large-Scale Scientific Computing*, pages 405–412. Springer, 2005.

[55] Edson Flórez, Carlos J Barrios, and Johnatan E Pecero. Methods for job scheduling on computational grids: Review and comparison. In *Latin American High Performance Computing Conference*, pages 19–33. Springer, 2015.

[56] Ian Foster and Carl Kesselman. *The grid : blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers, 1999.

[57] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier, 2003.

[58] Ian Foster and Carl Kesselman. The history of the grid. *computing*, 20(21):22, 2010.

[59] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International journal of high performance computing applications*, 15(3):200–222, 2001.

[60] Dorian Gaertner and Keith L Clark. On optimal parameters for ant colony optimization algorithms. In *IC-AI*, pages 83–89, 2005.

[61] Luca Maria Gambardella and Marco Dorigo. Has-sop: Hybrid ant system for the sequential ordering problem. Technical Report IDSIA 11-97, Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, Lugano, Switzerland, 1997.

[62] Yang Gao, Hongqiang Rong, and Joshua Zhexue Huang. Adaptive grid job scheduling with genetic algorithms. *Future Generation Computer Systems*, 21(1):151–161, 2005.

[63] Saurabh Kumar Garg, Rajkumar Buyya, and Howard Jay Siegel. Time and cost trade-off management for scheduling parallel applications on utility grids. *Future Generation Computer Systems*, 26(8):1344–1355, 2010.

[64] Farhad Soleimanian Gharehchopogh, Majid Ahadi, Isa Maleki, Ramin Habibpour, and Amin Kamalinia. Analysis of scheduling algorithms in grid computing environment. *ISSR Journals*, 2013.

[65] Christos Gogos, Christos Valouxis, Panayiotis Alefragis, George Goulas, Nikolaos Voros, and Efthymios Housos. Scheduling independent tasks on heterogeneous processors using heuristics and column pricing. *Future Generation Computer Systems*, 60:48–66, 2016.

[66] David E Goldberg, Robert Lingle, et al. Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[67] Martin Grajcar. Genetic list scheduling algorithm for scheduling and allocation on a loosely coupled heterogeneous multiprocessor system. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 280–285. IEEE, 1999.

[68] Martin Grajcar. Strength and weaknesses of genetic list scheduling for heterogeneous systems. In *acsd*, page 123. IEEE, 2001.

[69] Plerre-P Grass. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la thorie de la stigmergie: Essai d'interprtation du comportement des termites constructeurs. *Insectes sociaux*, 6(1): 4181, 1959.

[70] Thomas Grüninger and David Wallace. Multimodal optimization using genetic algorithms. *Master's thesis, Stuttgart University*, 1996.

[71] Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of job-scheduling strategies for grid computing. In *International Workshop on Grid Computing*, pages 191–202. Springer, 2000.

[72] Pierre Hansen, Nenad Mladenović, Raca Todosijević, and Saïd Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[73] Yongsheng Hao and Guanfeng Liu. Evaluation of nine heuristic algorithms with data-intensive jobs and computing-intensive jobs in a dynamic environment. *IET software*, 9(1):7–16, 2015.

[74] Tony Hey and Anne E Trefethen. The uk e-science core programme and the grid. In *International Conference on Computational Science*, pages 3–21. Springer, 2002.

[75] John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[76] Bert Hölldobler, Edward O Wilson, et al. *Journey to the ants: a story of scientific exploration.* Harvard University Press, 1994.

[77] Oscar H Ibarra and Chul E Kim. Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, 24(2):280–289, 1977.

[78] Hesam Izakian, Ajith Abraham, and Vaclav Snasel. Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In *Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on*, volume 1, pages 8–12. IEEE, 2009.

[79] Hesam Izakian, Ajith Abraham, and Vaclav Snasel. Performance comparison of six efficient pure heuristics for scheduling meta-tasks on heterogeneous distributed environments. *Neural Network World*, 19(6):695, 2009.

[80] Hesam Izakian, Behrouz Tork Ladani, Ajith Abraham, Vaclav Snasel, et al. A discrete particle swarm optimization approach for grid job scheduling. *International Journal of Innovative Computing, Information and Control*, 6(9):1–15, 2010.

[81] Laetitia Jourdan, Matthieu Basseur, and E-G Talbi. Hybridizing exact methods and metaheuristics: A taxonomy. *European Journal of Operational Research*, 199(3): 620–629, 2009.

[82] Leila Kallel and Marc Schoenauer. Alternative random initialization in genetic algorithms. In *ICGA*, pages 268–275, 1997.

[83] Sara Kardani-Moghaddam, Farzad Khodadadi, Reza Entezari-Maleki, and Ali Movaghar. A hybrid genetic algorithm and variable neighborhood search for task scheduling problem in grid environment. *Procedia Engineering*, 29:3808–3814, 2012.

[84] Ashfaq A. Khokhar, Viktor K. Prasanna, Muhammad E. Shaaban, and C-L Wang. Heterogeneous computing: Challenges and opportunities. *Computer*, 26(6):18–27, 1993.

[85] Joanna Kołodziej and Fatos Xhafa. Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. *Future Generation Computer Systems*, 27(8):1035–1046, 2011.

[86] Joanna Kołodziej and Fatos Xhafa. Integration of task abortion and security requirements in ga-based meta-heuristics for independent batch grid scheduling. *Computers & Mathematics with Applications*, 63(2):350–364, 2012.

[87] Joanna Kolodziej, Samee Ullah Khan, and Fatos Xhafa. Genetic algorithms for energy-aware scheduling in computational grids. In *2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, pages 17–24. IEEE, 2011.

[88] Joanna Kołodziej, Samee Ullah Khan, Lizhe Wang, Marek Kisiel-Dorohinicki, Sajjad A Madani, Ewa Niewiadomska-Szynkiewicz, Albert Y Zomaya, and Cheng-Zhong Xu. Security, energy, and performance-aware resource allocation mechanisms for computational grids. *Future Generation Computer Systems*, 31:77–92, 2014.

[89] K Kousalya and P Balasubramanie. To improve ant algorithm's grid scheduling using local search. *International Journal Of Computational Cognition http://www. yangsky. com/ijcc/pdf/ijcc747. pdf*, 7(4):47–57, 2009.

[90] Ku Ruhana Ku-Mahamud and Mustafa Muwafak Alobaedy. New heuristic function in ant colony system for job scheduling in grid computing. In *Proceeding of the 17th International Conference on Applied Mathematics. Montreux*, pages 47–52, 2012.

[91] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4): 406–471, 1999.

[92] Maozhen Li and Mark Baker. *The grid: core technologies*. John Wiley & Sons, 2005.

[93] Hongbo Liu, Ajith Abraham, and Aboul Ella Hassanien. Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. *Future Generation Computer Systems*, 26(8):1336–1343, 2010.

[94] Siriluck Lorpunmanee, Mohd Noor Sap, Abdul Hanan Abdullah, and Chai Chompoo-inwai. An ant colony optimization for dynamic job scheduling in grid environment. *International Journal of Computer and Information Science and Engineering*, 1(4):207–214, 2007.

[95] Heikki Maaranen, Kaisa Miettinen, and Antti Penttinen. On initial populations of a genetic algorithm for continuous optimization problems. *Journal of Global Optimization*, 37(3):405, 2007.

[96] Mojtaba MadadyarAdeh and Jamshid Bagherzadeh. An improved ant algorithm for grid scheduling problem using biased initial ants. In *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, volume 2, pages 373–378. IEEE, 2011.

[97] Maurice Maeterlinck. *Life of the white ant*. Dodd, Mead & Co., New York, 1927.

[98] Frédéric Magoulès. *Fundamentals of grid computing: theory, algorithms and technologies*. CRC Press, 2009.

[99] Frédéric Magoulès, Lei Yu, et al. *Grid resource management: toward virtual and services compliant grid computing*. CRC Press, 2008.

[100] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of parallel and distributed computing*, 59(2):107–131, 1999.

[101] Muthucumaru Maheswaran, Tracy D Braun, and Howard Jay Siegel. Heterogeneous distributed computing. *Wiley encyclopedia of electrical and electronics engineering*, 1999.

[102] JY Maipan-uku, I Rabiu, and Amit Mishra. Immediate/batch mode scheduling algorithms for grid computing: A review. *International Journal of Research - GRANTHAALAYAH*, 2017.

[103] Vittorio Maniezzo and Alberto Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge & Data Engineering*, 11(5): 769–778, 1999.

[104] EN Marais. Die siel van die mier (the soul of the ant). JL van Schaik, Pretoria, South Africa,, 1948 (first published in 1937).

[105] P Mathiyalagan, S Suriya, and SN Sivanandam. Modified ant colony algorithm for grid scheduling. *International Journal on computer science and Engineering*, 2(02): 132–139, 2010.

[106] Michalis Mavrovouniotis and Shengxiang Yang. Ant colony optimization with immigrants schemes in dynamic environments. In *International Conference on Parallel Problem Solving from Nature*, pages 371–380. Springer, 2010.

[107] Michalis Mavrovouniotis and Shengxiang Yang. Ant colony optimization with memory-based immigrants for the dynamic vehicle routing problem. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[108] Michalis Mavrovouniotis and Shengxiang Yang. Ant algorithms with immigrants schemes for the dynamic vehicle routing problem. *Information Sciences*, 294: 456–477, 2015.

[109] Michalis Mavrovouniotis and Shengxiang Yang. Empirical study on the effect of population size on max-min ant system in dynamic environments. In *Evolutionary Computation (CEC), 2016 IEEE Congress on*, pages 853–860. IEEE, 2016.

[110] Michalis Mavrovouniotis, Anastasia Ioannou, and Shengxiang Yang. Pre-scheduled colony size variation in dynamic environments. In *European Conference on the Applications of Evolutionary Computation*, pages 128–139. Springer, 2017.

[111] Michalis Mavrovouniotis, Felipe M Müller, and Shengxiang Yang. Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE transactions on cybernetics*, 47(7):1743–1756, 2017.

[112] Michalis Mavrovouniotis, Mien Van, and Shengxiang Yang. Pheromone modification strategy for the dynamic travelling salesman problem with weight changes. In *Computational Intelligence (SSCI), 2017 IEEE Symposium Series on*, pages 1–8. IEEE, 2017.

[113] Gabriela F Minetti, Gabriel Luque, and Enrique Alba. The problem aware local search algorithm: an efficient technique for permutation-based problems. *Soft Computing*, pages 1–14, 2017.

[114] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[115] Ehsan Ullah Munir, Jianzhong Li, Shengfei Shi, Zhaonian Zou, and Qaisar Rasool. A performance study of task scheduling heuristics in hc environment. In *Modelling, Computation and Optimization in Information Systems and Management Sciences*, pages 214–223. Springer, 2008.

[116] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[117] Sergio Nesmachnow, Enrique Alba, and Héctor Cancela. Scheduling in heterogeneous computing and grid environments using a parallel chc evolutionary algorithm. *Computational Intelligence*, 28(2):131–155, 2012.

[118] Sergio Nesmachnow, Héctor Cancela, and Enrique Alba. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Applied Soft Computing*, 12(2):626–639, 2012.

[119] Sigurdur Ólafsson. Metaheuristics. *Handbooks in operations research and management science*, 13:633–654, 2006.

[120] IM Oliver, DJd Smith, and John RC Holland. Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987., 1987.

[121] E Osaba, R Carballedo, F Diaz, E Onieva, I de la Iglesia, and A Perallos. Crossover versus mutation: a comparative analysis of the evolutionary strategy of genetic algorithms applied to combinatorial optimization problems. *The Scientific World Journal*, 2014, 2014.

[122] Oshin and Mahesh Chandra Bhatt. Hybrid pso for job scheduling to minimize makespan in heterogeneous grids. In *Communication and Electronics Systems (ICCES), 2017 2nd International Conference on*, pages 586–591. IEEE, 2017.

[123] Oshin and Amit Chhabra. Job scheduling using ant colony optimization in grid environment. In *Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on*, pages 2845–2850. IEEE, 2016.

[124] Elina Pacini, Cristian Mateos, and Carlos García Garino. Distributed job scheduling based on swarm intelligence: A survey. *Computers & Electrical Engineering*, 40 (1):252–269, 2014.

[125] Elina Pacini, Cristian Mateos, Carlos García Garino, Claudio Careglio, and Aníbal Mirasso. A bio-inspired scheduler for minimizing makespan and flowtime of computational mechanics applications on federated clouds. *Journal of Intelligent & Fuzzy Systems*, 31(3):1731–1743, 2016.

[126] Andrew J Page and Thomas J Naughton. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *Artificial Intelligence Review*, 24(3-4):415–429, 2005.

[127] Manish Parashar and Craig A Lee. Special issue on grid computing. *Proceedings of the IEEE*, 93(3):479–484, 2005.

[128] Radu Prodan and Thomas Fahringer. Zenturio: a grid middleware-based tool for experiment management of parallel and distributed applications. *Journal of Parallel and Distributed Computing*, 64(6):693–707, 2004.

[129] Radu Prodan and Thomas Fahringer. Dynamic scheduling of scientific workflow applications on the grid: a case study. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 687–694. ACM, 2005.

[130] Arun K Pujari. *Data mining techniques*. Universities press, 2001.

[131] Marjan Kuchaki Rafsanjani and Amid Khatibi Bardsiri. A new heuristic approach for scheduling independent tasks on heterogeneous computing systems. *International Journal of Machine Learning and Computing*, 2(4):371, 2012.

[132] V Rajaraman. Grid computing. *Resonance*, 21(5):401–415, 2016.

[133] Naglaa M Reda, A Tawfik, Mohamed A Marzok, and Soheir M Khamis. Sort-mid tasks scheduling algorithm in grid computing. *Journal of advanced research*, 6(6): 987–993, 2015.

[134] Graham Ritchie. Static multi-processor scheduling with ant colony optimisation & local search. Master's thesis, Citeseer, 2003.

[135] Graham Ritchie and John Levine. A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In *23rd Workshop UK Planning and Scheduling Special Interest Group (PlanSIG 2004)*. AAAI, 2004.

[136] Kumara Sastry, David Goldberg, and Graham Kendall. Genetic algorithms. In *Search methodologies*, pages 97–125. Springer, 2005.

[137] S Selvi and D Manimegalai. Multiobjective variable neighborhood search algorithm for scheduling independent jobs on computational grid. *Egyptian Informatics Journal*, 16(2):199–212, 2015.

[138] S Selvi and D Manimegalai. Task scheduling using two-phase variable neighborhood search algorithm on heterogeneous computing and grid environments. *Arabian Journal for Science & Engineering (Springer Science & Business Media BV)*, 40(3), 2015.

[139] S Selvi, D Manimegalai, and A Suruliandi. Efficient job scheduling on computational gridwith differential evolution algorithm. *International Journal of Computer Theory and Engineering*, 3(2):277, 2011.

[140] Pankaj Shroff, Daniel W Watson, Nicholas S Flann, and Richard F Freund. Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *5th Heterogeneous Computing Workshop (HCW'96)*, pages 98–117, 1996.

[141] Howard Jay Siegel, Henry G Dietz, and John K Antonio. Software support for heterogeneous computing. *ACM Computing Surveys (CSUR)*, 28(1):237–239, 1996.

[142] C Sriskandarajah, AKS Jardine, and CK Chan. Maintenance scheduling of rolling stock using a genetic algorithm. *Journal of the Operational Research Society*, 49 (11):1130–1145, 1998.

[143] T Stützle. *Local search algorithms for combinatorial problems— analysis, improvements, and new applications*. PhD thesis, Department of Computer Science, Darmstadt University of Technology, Darmstadt, Germany, 1998.

[144] Thomas Stutzle and Holger Hoos. Max-min ant system and local search for the traveling salesman problem. In *Evolutionary Computation, 1997., IEEE International Conference on*, pages 309–314. IEEE, 1997.

[145] Thomas Stützle and Holger Hoos. The max-min ant system and local search for combinatorial optimization problems. In *Meta-heuristics*, pages 313–329. Springer, 1999.

[146] Thomas Stützle and Holger H Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.

[147] Thomas Stützle, Manuel López-Ibánez, Paola Pellegrini, Michael Maur, Marco Montes De Oca, Mauro Birattari, and Marco Dorigo. Parameter adaptation in ant colony optimization. In *Autonomous search*, pages 191–215. Springer, 2011.

[148] Prasanna Sugavanam, Howard Jay Siegel, Anthony A Maciejewski, Mohana Oltikar, Ashish Mehta, Ron Pichel, Aaron Horiuchi, Vladimir Shestak, Mohammad Al-Otaibi, Yogish Krishnamurthy, et al. Robust static allocation of resources for independent tasks under makespan and dollar cost constraints. *Journal of Parallel and Distributed Computing*, 67(4):400–416, 2007.

[149] Prasanna V Sugavanam, Howard Jay Siegel, Anthony A Maciejewski, Syed Amjad Ali, Mohammad Al-Otaibi, Mahir Aydin, Kumara Guru, Aaron Horiuchi, Yogish G Krishnamurthy, Panho Lee, et al. Processor allocation for tasks that is robust against errors in computation time estimates. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 14–pp. IEEE, 2005.

[150] Gilbert Syswerda. Uniform crossover in genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 2–9. Morgan Kaufmann Publishers, 1989.

[151] Arne Thesen. Design and evaluation of tabu search algorithms for multiprocessor scheduling. *Journal of Heuristics*, 4(2):141–160, 1998.

[152] Mitchell D Theys, Tracy D Braun, HJ Siegal, Anthony A Maciejewski, and YK Kwok. Mapping tasks onto distributed heterogeneous computing systems using a genetic algorithm approach. *Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences*, pages 135–178, 2001.

[153] D Thilagavathi and Antony Selvadoss Thanamani. A survey on dynamic job scheduling in grid environment based on heuristic algorithms. *arXiv preprint arXiv:1402.5205*, 2014.

[154] YG Tirat-Gefen and Alice C Parker. Mega: An approach to system-level design of application specific heterogeneous multiprocessors. In *Proc. Heterogeneous Comput. Workshop, Int. Parallel Process. Symp*, volume 105, 1996.

[155] SG Tzafestas, M-P Saltouros, and M Markaki. A tutorial overview of genetic algorithms and their applications. In *Soft Computing in Systems and Control Technology*, pages 223–300. World Scientific, 1999.

[156] Lee Wang, Howard Jay Siegel, Vwani P Roychowdhury, and Anthony A Maciejewski. Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of parallel and distributed computing*, 47(1):8–22, 1997.

[157] Meihong Wang and Wenhua Zeng. A comparison of four popular heuristics for task scheduling problem in computational grid. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on*, pages 1–4. IEEE, 2010.

[158] L Darrell Whitley et al. The genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *ICGA*, volume 89, pages 116–123. Fairfax, VA, 1989.

[159] F Xhafa and J Carretero. Experimental study of ga-based schedulers in dynamic distributed computing environments. In *Optimization Techniques for Solving Complex Problems*, pages 423–441. Wiley, 2009.

[160] Fatos Xhafa. A hybrid evolutionary heuristic for job scheduling on computational grids. In *Hybrid Evolutionary Algorithms*, pages 269–311. Springer, 2007.

[161] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[162] Fatos Xhafa and Ajith Abraham. Computational models and heuristic methods for grid scheduling problems. *Future generation computer systems*, 26(4):608–621, 2010.

[163] Fatos Xhafa, Leonard Barolli, and Arjan Durresi. An experimental study on genetic algorithms for resource allocation on grid systems. *Journal of Interconnection Networks*, 8(04):427–443, 2007.

[164] Fatos Xhafa, Javier Carretero, Leonard Barolli, and Arjan Durresi. Requirements for an event-based simulation package for grid systems. *Journal of Interconnection Networks*, 8(02):163–178, 2007.

[165] Fatos Xhafa, Enrique Alba, Bernabé Dorronsoro, Bernat Duran, and Ajith Abraham. Efficient batch job scheduling in grids using cellular memetic algorithms. In *Metaheuristics for Scheduling in Distributed Computing Environments*, pages 273–299. Springer, 2008.

[166] Fatos Xhafa, Bernat Duran, Ajith Abraham, and Keshav P Dahal. Tuning struggle strategy in genetic algorithms for scheduling in computational grids. In *Proceedings of the 2008 7th Computer Information Systems and Industrial Management Applications*, pages 275–280. IEEE Computer Society, 2008.

[167] Fatos Xhafa, Juan Antonio Gonzalez, Keshav P Dahal, and Ajith Abraham. A ga (ts) hybrid algorithm for scheduling in computational grids. *HAIS*, 9:285–292, 2009.

[168] Fatos Xhafa, Bernat Duran, Joanna Kolodziej, Leonard Barolli, and Makoto Takizawa. On exploitation vs exploration of solution space for grid scheduling. In *2011 Third International Conference on Intelligent Networking and Collaborative Systems*, pages 164–171. IEEE, 2011.

[169] Fatos Xhafa, Joanna Kolodziej, Leonard Barolli, and Akli Fundo. A ga+ ts hybrid algorithm for independent batch scheduling in computational grids. In *Network-Based Information Systems (NBiS), 2011 14th International Conference on*, pages 229–235. IEEE, 2011.

[170] Fatos Xhafa, Joanna Kolodziej, Leonard Barolli, Vladi Kolici, Rozeta Miho, and Makoto Takizawa. Evaluation of hybridization of ga and ts algorithms for independent batch scheduling in computational grids. In *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2011 International Conference on*, pages 148–155. IEEE, 2011.

[171] Fatos Xhafa, Javier Carretero, Bernabé Dorronsoro, and Enrique Alba. A tabu search algorithm for scheduling independent jobs in computational grids. *Computing and informatics*, 28(2):237–250, 2012.

[172] Hong Li Yin and Yong Ming Wang. Genetic algorithm with parameters optimization mechanism for hard scheduling problems. In *Computer and Information Technology (CIT), 2012 IEEE 12th International Conference on*, pages 587–591. IEEE, 2012.

[173] Muhanad Tahrir Younis and Shengxiang Yang. A genetic algorithm for independent job scheduling in grid computing. *MENDEL Soft Computing Journal*, 23(01):65–72, 2017.

[174] Muhanad Tahrir Younis and Shengxiang Yang. Hybrid meta-heuristic algorithms for independent job scheduling in grid computing. *Applied Soft Computing*, 2018.

[175] Muhanad Tahrir Younis, Shengxiang Yang, and Benjamin Passow. Meta-heuristically seeded genetic algorithm for independent job scheduling in grid computing. In *European Conference on the Applications of Evolutionary Computation*, pages 177–189. Springer, 2017.

[176] Muhanad Tahrir Younis, Shengxiang Yang, and Benjamin N Passow. A loosely coupled hybrid meta-heuristic algorithm for the static independent task scheduling problem in grid computing. In *IEEE World Congress on Computational Intelligence*, pages 1746–1753. IEEE Press, 2018.

[177] Shirin Dehghani Zahedani and GholamHossin Dastghaibyfard. A hybrid batch job scheduling algorithm for grid environment. In *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*, pages 763–768. IEEE, 2014.

[178] Yanmin Zhu and Lionel M Ni. A survey on grid scheduling systems. *Department of Computer Science, Hong Kong University of science and Technology*, 32:1–47, 2003.

[179] Albert Y. Zomaya and Yee-Hwei Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE transactions on parallel and distributed systems*, 12 (9):899–911, 2001.